

INTERNATIONAL MASTER OF COMPUTATIONAL MECHANICS

UNIVERSITY OF DUISBURG-ESSEN

Master's Thesis in Computational Mechanics

Investigating the Impact of Randomness in Reproducibility in Computer Vision

A Study on Applications in Civil Engineering and Medicine

Bahadir Eryilmaz

Master Thesis

Investigating the Impact of Randomness in Reproducibility in Computer Vision

A Study on Applications in Civil Engineering and Medicine

Faculty of Engineering
University of Duisburg-Essen

Bahadir Eryilmaz
M.Sc. Computational Mechanics
Matriculation Number: 3125111
Submitted: 13.09.2023

Supervisor: Prof. Dr.-Ing. Carolin Birk
Second Reviewer: Dr. Mana Mohajer
External Supervisor: Prof. Dr. Christin Seifert
External Daily Supervisor: M.Sc. Osman Alperen Koras
The Institute for AI in Medicine
Multimodal Computing and Machine Intelligence (MCMI)

Declaration

I hereby declare that this master's thesis is my original work, and I have appropriately acknowledged and cited all sources and materials used.

Location: Essen

Date: 13.09.2023

Signature: _____

Name: Bahadir Eryilmaz

Acknowledgments

First and foremost, I would like to express my profound gratitude to Prof. Dr. Christin Seifert, whose invaluable guidance and visionary ideas laid the foundation for this thesis. Her unwavering support and insightful directions have been instrumental in shaping the trajectory of this research.

I am deeply thankful to Prof. Dr.-Ing. habil. Carolin Birk for her official internal supervision. Her trust in my abilities provided the confidence and framework necessary for me to embark on this academic journey.

Special thanks are due to Dr. Jörg Schlötterer for his invaluable assistance in crafting the experimental design. His expertise and insights were crucial in ensuring the robustness of our research methodology.

I owe a debt of gratitude to M.Sc. Osman Alperen Koras, my daily supervisor, whose influence permeates every facet of this thesis. His consistent guidance, support, and constructive feedback have been indispensable in navigating the complexities of this research.

I am also grateful to M.Sc. Van Bach Nguyen for his expertise and guidance during the initial phases of the thesis. His direction was crucial in setting the right course for the research.

My sincere appreciation goes to M.Sc. Shreyasi Pathak for her assistance on some of the most critical aspects of the thesis. Her expertise played a key role in advancing the experiments and ensuring their success.

Furthermore, I would like to acknowledge the Institute for AI in Medicine for providing access to their computer cluster. The experiments conducted for this thesis greatly benefited from their state-of-the-art hardware and facilities. Furthermore, I extend my heartfelt gratitude to the entire team at the Institute of AI in Medicine for their unwavering support and assistance throughout my time there. Their guidance and friendship have enriched my academic experience.

Lastly, I wish to express my thanks to the broader scientific community for their continuous efforts in advancing knowledge. Their dedication serves as an inspiration and has been the bedrock upon which this research stands.

Abstract

Reproducibility is a fundamental requirement for scientific research, yet it remains challenging in computer vision due to various sources of randomness, notably CUDA. This parallel computing platform, while enabling high-performance execution of computer vision algorithms on GPUs, does not guarantee deterministic outcomes across runs. In this thesis, we investigate the impact of CUDA-induced randomness on reproducibility across diverse datasets: CIFAR for image classification, a civil engineering dataset for concrete crack detection, and a medical dataset for breast cancer diagnosis. By controlling other potential sources of variability, such as weight initialization, data shuffling, and data augmentation, we isolate the effects of CUDA randomness and compare outcomes across multiple runs, both with and without deterministic CUDA execution. Another aspect of our research involves analyzing the weights of the models, providing deeper insights into the internal dynamics and further substantiating our claims. We also evaluate the trade-offs between reproducibility and performance when opting for deterministic CUDA execution. In line with responsible research, we report the environmental impact of our experiments, highlighting the carbon footprint in a region reliant on fossil-derived electricity. This thesis will contribute to the advancement of reproducible research in computer vision by providing a systematic and comprehensive evaluation of CUDA randomness and its effects on different computer vision tasks and domains.

Contents

List of Figures

List of Tables

Abbreviations

1 Introduction

Deep learning is a branch of machine learning that uses neural networks to learn from data and perform tasks such as computer vision, which is the field of study that enables machines to understand and analyze visual information such as images and videos. Computer vision algorithms analyze certain criteria in images and videos, and then apply interpretations to predictive or decision-making tasks. Most recent computer vision algorithms are deep learning based, as they can leverage the power of multiple layers of linear and non-linear operations to extract complex features from the input data. Deep learning algorithms use back propagation and weight optimization to adjust the network parameters based on the error between the predicted and actual outputs [1]. However, deep learning and computer vision are not deterministic fields [2]. There are various sources of randomness and irreproducibility that can affect the performance and reliability of deep learning models and computer vision algorithms.

1.1 Randomness in Deep Learning and its Impact on Reproducibility

Randomness plays a crucial role in deep learning world, directly influencing the reproducibility of results. It introduces an element of unpredictability into the training process of neural networks. When randomness is at play, identical input conditions can yield varying outcomes. Such variability necessitates a thorough examination of randomness to ensure reproducibility in machine learning endeavors.

In an ideal world, we would want deep learning algorithms to be stable, producing consistent results every time they are run with the same inputs. This stability would make it easier to compare different models, verify results, and build upon previous work. However, in practice, the inherent randomness in many aspects of deep learning—from the initialization of weights to the shuffling of training data—means that even with identical input conditions, we can observe different outcomes. This unpredictability is not just a theoretical concern. In real-world applications, it can lead to significant differences in performance, making it challenging to determine whether a new model or technique is genuinely better or just benefited from a lucky random seed.

In the context of this thesis, we embrace the definition of reproducibility posited by Goodman et al. [3], which emphasizes the ability to replicate results precisely by employing identical data and tools. Reproducibility stands as a foundational pillar of the scientific method. It's paramount in validating results in computer vision tasks, especially when these results influence critical decisions, be it medical diagnoses or assessing the reliability of a structure. Furthermore, reproducibility can streamline computer vision tasks by obviating the need for redundant experiments to validate findings.

However, lapses in reproducibility within computer vision can erode trust in research outcomes and diminish the perceived reliability of findings. To bolster the credibility of novel methodologies, it's imperative for researchers to be transparent, sharing essential resources like datasets, trained models, training parameters, and evaluation scripts in their publications [4]. Such transparency facilitates replication of experiments by peers, fostering validation and furthering progress in the domain. Yet, even with these measures, achieving reproducibility can be fraught with challenges due to myriad factors [5].

Randomness emerges as a primary culprit behind irreproducibility. While there are various sources of randomness and irreproducibility, this work zeroes in on the randomness inherent in deep neural networks. Such randomness can be categorized into two primary types: implementation-level randomness and algorithmic-level randomness [2] which we will look into later. In deep learning algorithms, randomness is sometimes deliberately introduced to deter the algorithm from merely memorizing data. However, as highlighted earlier, the implications of unchecked randomness can be detrimental.

1.2 The Pervasiveness of GPUs in Deep Learning and the Imperative for Randomness Investigation

In the deep learning, the utilization of GPU! (GPU!) has become increasingly prevalent due to their inherent advantages in handling the computational demands of complex neural networks. As delineated in a study [6] GPUs are particularly adept at managing the large matrices inherent in deep learning tasks, especially in the context of CNN! (CNN!) [7]. The study showcased that GPU! implementations not only outperformed their CPU! (CPU!) counterparts in terms of execution speed but also exhibited superior scalability with increasing network sizes and input dimensions.

The CUDA! (CUDA!) [8] framework by NVIDIA, a proprietary platform tailored for GPU! programming, further facilitates this by enabling efficient multithreading and access to diverse memory types on the GPU!. Such computational prowess underscores the rationale behind the growing preference for GPUs in deep learning endeavors, as they offer both speed and efficiency in training and inference tasks. Hence, it is crucial to investigate randomness caused by CUDA execution, given its widespread use in deep learning community.

In this work, we embark on a meticulous investigation into the randomness caused by CUDA execution. By maintaining a controlled environment and ensuring deterministic CUDA operations, we aim to delve deep into the trade-offs that emerge. Our approach is holistic: we not only probe the randomness but also assess its tangible impact on experiments conducted across three distinct datasets. It's anticipated that varying settings will yield disparate performance metrics, shedding light on the intricate interplay between task performance, runtime, and computational overheads. This exploration serves as a precursor to a more in-depth analysis, setting the stage for understanding the broader implications of CUDA randomness in deep learning endeavors.

1.3 Problem Statement and Research Objectives

In the domain of deep learning, particularly within computer vision tasks, the reproducibility of experiments is paramount. However, the inherent non-deterministic nature introduced by CUDA, a parallel computing platform, poses challenges to achieving consistent and reproducible outcomes across different runs. This unpredictability, stemming from CUDA's execution, can compromise the reliability and validity of computer vision algorithms, making it imperative to investigate and understand the depth of its impact. The research objectives can be summarized as follows:

1. **Empirical Examination:** To conduct a thorough empirical analysis on datasets such as CIFAR for image classification, a civil engineering dataset for concrete crack detection, and a medical dataset for breast cancer diagnosis.
2. **Isolation of CUDA's Impact:** To isolate and quantify the effects of CUDA randomness by controlling other potential sources of variability in computer vision experiments.
3. **Deterministic vs. Non-deterministic CUDA Execution:** To systematically compare the outcomes of computer vision algorithms under both deterministic and non-deterministic CUDA settings.
4. **Analysis and Discussion:** To evaluate and discuss the results, drawing insights and implications from the findings.
5. **Establishment of Guidelines:** To formulate guidelines based on the research findings, aiding practitioners in managing and mitigating the effects of CUDA randomness in deep learning tasks.

1.4 Research Questions

From the research objectives, we can formulate a main research question and three sub-questions as follows:

MQ: *What is the overarching impact of CUDA randomness on the reproducibility and performance of deep learning tasks in computer vision?*

1. **SQ1:** What is the extent of performance variability when controlling for other sources of randomness, while allowing for randomness from CUDA execution to be present?
2. **SQ2:** What is the cost of using deterministic approaches in CUDA randomness?
3. **SQ3:** How does the randomness in Computer Vision impact the task performance and computation cost for specific applications, such as Civil Engineering and Medicine?

1.5 Structure

Starting with **Background**, this section establishes a strong foundation by explaining the important background and theoretical concepts. It ensures that readers have a clear understanding of the context, making subsequent sections more accessible and meaningful.

Following that, the **Literature Review** delves into the big landscape of prior work, spotlighting relevant studies and experimental findings. By understanding the existing body of knowledge, we can better appreciate the novelty and significance of the presented research.

The **Research Methodology** illuminates the chosen research approach, guiding readers through the methodology and design principles of the experiments. This section emphasizes the rigor and credibility of the conducted research, allowing for reproducibility and validation by peers.

In the **Experiment Results**, there's a transparent showcase of the raw outcomes from the experiments. All findings are summarized and presented in a meaningful manner. Observations are made to provide context, allowing readers to form their own interpretations based on the presented data.

The **Environmental Impact** addresses the increasingly crucial aspect of ecological responsibility. In a world grappling with environmental challenges, it's imperative to understand and report how the experiments align with sustainable practices, particularly in terms of power consumption.

The **Discussion** provides the interpretations to extrapolate the broader implications of the results. This section aims to connect the dots, pinpointing key takeaways, acknowledging the study's limitations, and opening the way for further research avenues that could potentially refine or expand upon the findings.

Finally, the **Conclusion** summarizes the main points of the thesis. It offers a concise recap of the findings and suggests how they might be applied in practical scenarios.

2 Background

This chapter provides the foundational concepts and underlying principles of the research. It delves into the foundational concepts of Computer Vision and Deep Learning, illuminating on specific algorithms and their underlying principles. The importance of reproducibility in scientific research is highlighted, with a particular focus on the challenges introduced by randomness in deep learning. This section further dissects the sources of this randomness, emphasizing the role of floating-point arithmetic. To provide a practical context, an overview of the various methods and tools utilized in this research, including optimizers, schedulers, and tracking tools, is presented. The aim is to ensure that readers, even those with a peripheral association with the field, can grasp the technical concepts and terminologies used throughout the thesis.

2.1 Deep Learning

2.1.1 History of Rise of AI and Deep Learning

Artificial Intelligence (AI) and its specialized branch, Deep Learning, trace their roots back to 1943 when Walter Pitts and Warren McCulloch introduced a model based on human neural networks. This foundation led to the introduction of backpropagation in the 1960s and the development of convolutional neural networks by Kunihiko Fukushima in the 1970s. Despite experiencing two significant setbacks known as the AI winters in the 1970s and late 1980s, the field saw major breakthroughs with the advent of support vector machines and LSTM networks.

The 21st century marked a transformative era for Deep Learning. Challenges like the Vanishing Gradient Problem were addressed, and the launch of the ImageNet database in 2009 catalyzed advancements in image recognition. By 2011, with faster GPUs, architectures like AlexNet emerged, outperforming traditional methods in international competitions. The pursuit of unsupervised learning was exemplified by Google Brain's Cat Experiment in 2012. The inception of the Generative Adversarial Neural Network (GAN) in 2014 further showcased the potential and versatility of deep neural networks.

The resurgence and widespread adoption of Deep Learning in the 21st century can be attributed to two key factors: Big Data and computational power. The explosion of Big Data came first, announced by the onset of the digital age and the accumulation of massive datasets. This was closely followed by advancements in computational power, particularly the capabilities of GPUs, which made it feasible to process these large datasets. The synergy of vast amounts of data and the capability to process it efficiently enabled the Deep Learning algorithms, which had been conceptualized decades ago, to finally deliver on their potential and gain immense popularity.

Table 2.1: Milestones in the Evolution of Neural Networks and AI

Year	Milestone
1943	Introduction of Neural Network Model by Pitts & McCulloch
1960s	Development of Backpropagation
1970s	First AI Winter & Introduction of Convolutional Neural Networks by Fukushima
1985-1990s	Second AI Winter & Advent of Support Vector Machines and LSTM
Early 2000s	Onset of the Big Data Era
2009	Launch of ImageNet
Late 2000s	Exponential growth in GPU capabilities
2011	Emergence of AlexNet
2012	Google Brain's Cat Experiment
2014	Introduction of Generative Adversarial Networks (GANs) by Ian Goodfellow

2.1.2 Advancements and Significance of Deep Learning in the Modern Technological Landscape

Deep learning, a subset of machine learning, has firmly established itself as one of the top technological advancements of the modern era. This technology, by emulating the structure and functionalities of the human brain through complex artificial neural networks, enables machines to learn and make independent decisions by processing vast data troves.

In traditional machine learning, algorithms heavily depended on manual feature extraction. This means that before data could be used in a model, its most relevant parts had to be pre-identified by humans. This manual identification was not only time-consuming but also prone to biases and oversights, limiting the potential and scalability of the models. In contrast, deep learning models thrive on automation, eliminating this manual process. By utilizing multi-layered neural networks, these models intuitively identify and extract fundamental features from input data. This unparalleled capability has birthed innovations and breakthroughs across numerous domains, from image and speech recognition to more sophisticated applications like medical diagnosis and natural language processing.

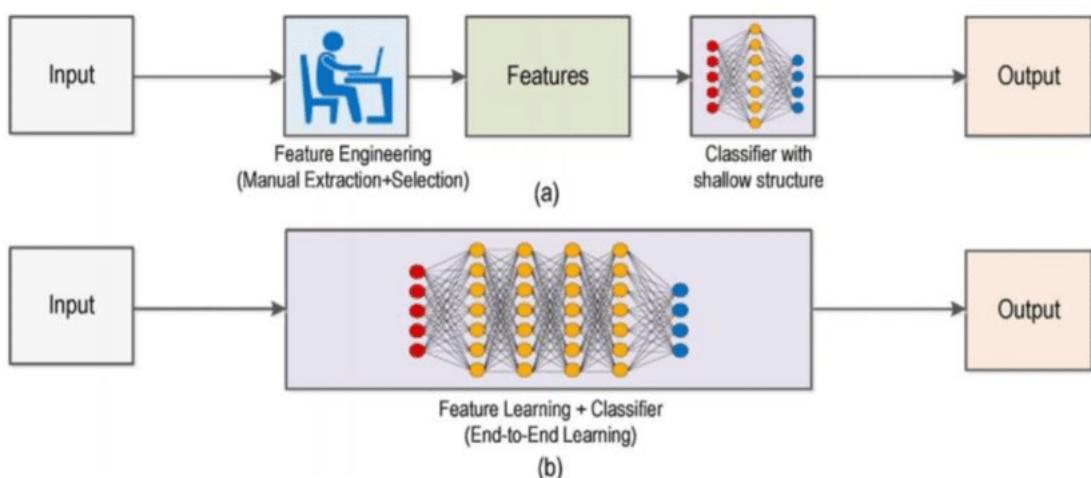


Figure 2.1: Side-by-side comparison of traditional machine learning (with manual feature extraction) and deep learning (automatic feature extraction) Source: [9].

The advancement of automation has only bolstered Deep Learning's relevance. In diverse applications, from robotic processes to customer service automation and even the intricate algorithms guiding self-driving vehicles, deep learning acts as the backbone. It provides these systems the tools to comprehend their environment and subsequently make informed decisions. This integration ensures not only unprecedented speed and efficiency but also a level of reliability and precision often surpassing human-driven processes.

Another frontier that deep learning is significantly influencing is Industry 4.0. This term, synonymous with the Fourth Industrial Revolution, marks the ongoing transformation characterized by heightened automation and data exchange in manufacturing technologies. It encompasses innovations like cyber-physical systems, the ever-expanding Internet of Things (IoT), cloud and cognitive computing. Within this ecosystem, deep learning emerges as a game-changer. Imagine a manufacturing setup where sensors constantly relay data from equipment. Deep learning models, acting on this data, can preemptively detect when a component might fail, ensuring timely maintenance, drastically reducing downtime, and invariably enhancing production efficiency.

To encapsulate, the significance of deep learning in our evolving technological landscape is huge. Whether it's in the seemingly mundane, like fine-tuning song or movie recommendations, or in the deeply impactful, such as diagnosing ailments from medical images or forecasting calamities, deep learning prevails. As the world becomes increasingly data-centric, the unparalleled processing and analytical capabilities of deep learning models ensure that they remain not just relevant, but indispensable. In many ways, deep learning is not merely a trend within artificial intelligence; it represents a bold stride towards devising machines that think and learn akin to humans, yet operate on scales unfathomable to human cognition.

2.1.3 Deep Learning Essentials

Deep learning, a subset of machine learning, has become a driving force behind various state-of-the-art applications in numerous domains. The following core concepts elucidate the foundational pillars of deep learning, providing insights into its inner workings and methodologies.

Neural Networks

Neural networks are the foundational building blocks of deep learning. They are inspired by the structure and functioning of the human brain's interconnected neurons. The concept of neural networks has gained immense popularity due to their ability to learn complex patterns and representations from data. By simulating the interconnectedness of neurons, neural networks can capture hierarchical features in data, making them suitable for tasks such as image recognition, natural language processing, and more. The layers in a neural network (input, hidden, and output) allow for the extraction of progressively abstract features from the input data, enabling the model to make accurate predictions.

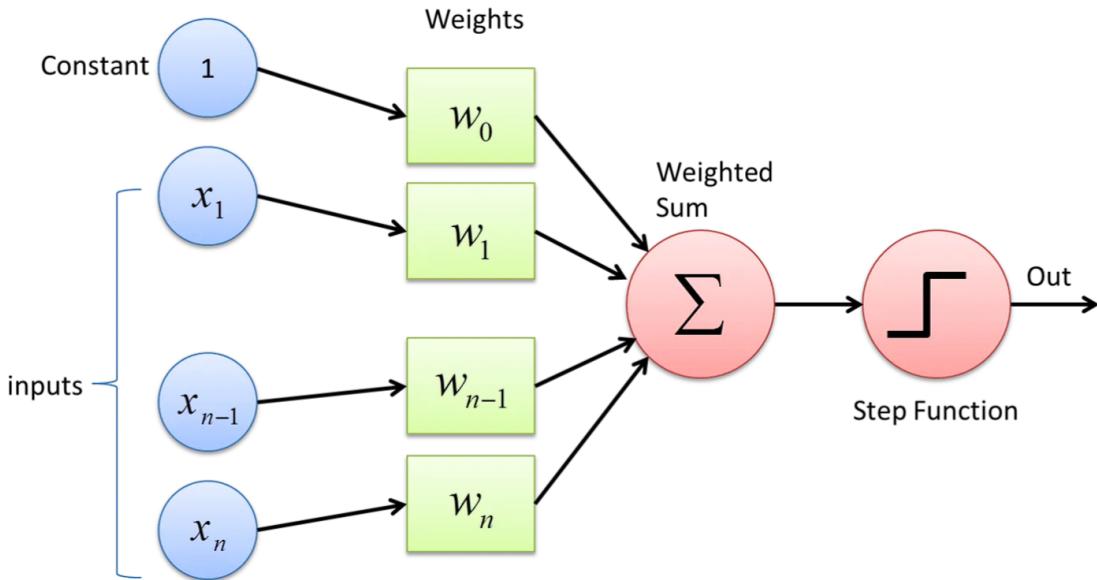


Figure 2.2: A perceptron, the basic building block of neural networks. Credits: Towards Data Science

Activation Functions

Activation functions introduce non-linearity to the neural network, allowing it to approximate and learn complex relationships in data. Without activation functions, neural networks would only be capable of representing linear transformations, severely limiting their expressive power. ReLU, Sigmoid, and Tanh are commonly used activation functions, each with its unique properties. ReLU addresses the vanishing gradient problem, Sigmoid maps inputs to a sigmoid-shaped range suitable for binary classification, and Tanh is often used in hidden layers to map inputs to a range between -1 and 1. The mathematical expressions for these activation functions and in Figure ?? a graph of these functions are presented below.

- **ReLU (Rectified Linear Unit):**

$$f(x) = \max(0, x)$$

- **Sigmoid:**

$$f(x) = \frac{1}{1 + e^{-x}}$$

- **Tanh:**

$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

Loss Functions

Loss functions quantify the difference between predicted and true values, providing a measure of how well the model is performing. They serve as the basis for optimization, helping the model adjust its parameters to minimize the discrepancy between predictions and ground truth. Different loss functions are used for different tasks; Mean Squared Error (MSE) is well-suited for regression tasks, while Cross-Entropy is commonly used for classification tasks. The choice of the appropriate loss function depends on the nature of the problem the neural network is tackling.

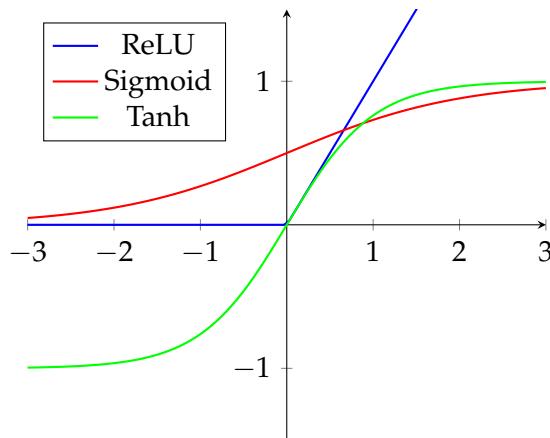


Figure 2.3: Visualization of Activation Functions

- **Mean Squared Error (MSE)** for regression tasks:

$$L(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

MSE is used for regression tasks where we predict continuous values. It's chosen because it penalizes larger errors more, it's differentiable for efficient optimization, and its convex nature aids in finding optimal solutions.

- **Cross-Entropy** for classification:

$$L(y, \hat{y}) = - \sum_i y_i \log(\hat{y}_i)$$

Cross-Entropy is for classification tasks. It's based on probabilities, making it suitable for class probabilities interpretation. Its logarithmic nature amplifies differences between predicted and true class probabilities, aiding optimization. Also, its gradient doesn't saturate, ensuring continuous learning.

Optimizers

Optimizers play a crucial role in training neural networks by adjusting the model's parameters to minimize the loss function. Gradient Descent and its variants, such as Stochastic Gradient Descent (SGD), SGD with Momentum, and Adam, are pivotal in this endeavor. These optimizers iteratively modify the model parameters, aiming to find the minima of the loss function, which in turn enhances the model's performance.

- **Gradient Descent (GD):** The algorithm aims to find the minimum of a function F by iteratively moving in the direction of steepest descent, as defined by the negative of the gradient ∇F at the current point \mathbf{x}_n . The update rule is given by

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n), \quad n \geq 0$$

where γ_n is the learning rate at iteration n .

- **Stochastic Gradient Descent (SGD):** Unlike GD, SGD approximates the gradient using a single training example, which can lead to faster convergence. The update for each parameter w is

$$w := w - \eta \nabla Q_i(w)$$

where η is the learning rate.

- **SGD with Momentum:** This variant accelerates SGD by adding a momentum term, which serves to dampen oscillations and speed up convergence. The update rule is:

$$v_t = \mu v_{t-1} - \eta \nabla Q_i(w)$$

$$w := w + v_t$$

where μ is the momentum term. (See *On the importance of initialization and momentum in deep learning* by Sutskever et al., [10] for more details.).

- **Adam:** The update rule for the parameters w in Adam is defined as

$$w_t = w(t-1) - \alpha \cdot \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \right),$$

where w_t and $w(t-1)$ are the parameter vectors at time t and $t-1$ respectively, α is the learning rate, \hat{m}_t and \hat{v}_t are bias-corrected estimates of the first and second moment, and ϵ is a small constant to prevent division by zero.

Backpropagation

Backpropagation is a fundamental algorithm for training neural networks. It is vital because it enables neural networks to learn from data by adjusting their weights and biases. The process involves calculating the gradients of the loss function with respect to each parameter using the chain rule from calculus. These gradients guide the optimization algorithm in adjusting the model's parameters to minimize the prediction error. Backpropagation is crucial for enabling neural networks to learn complex relationships in data, as it iteratively fine-tunes the model's parameters based on the feedback provided by the gradients.

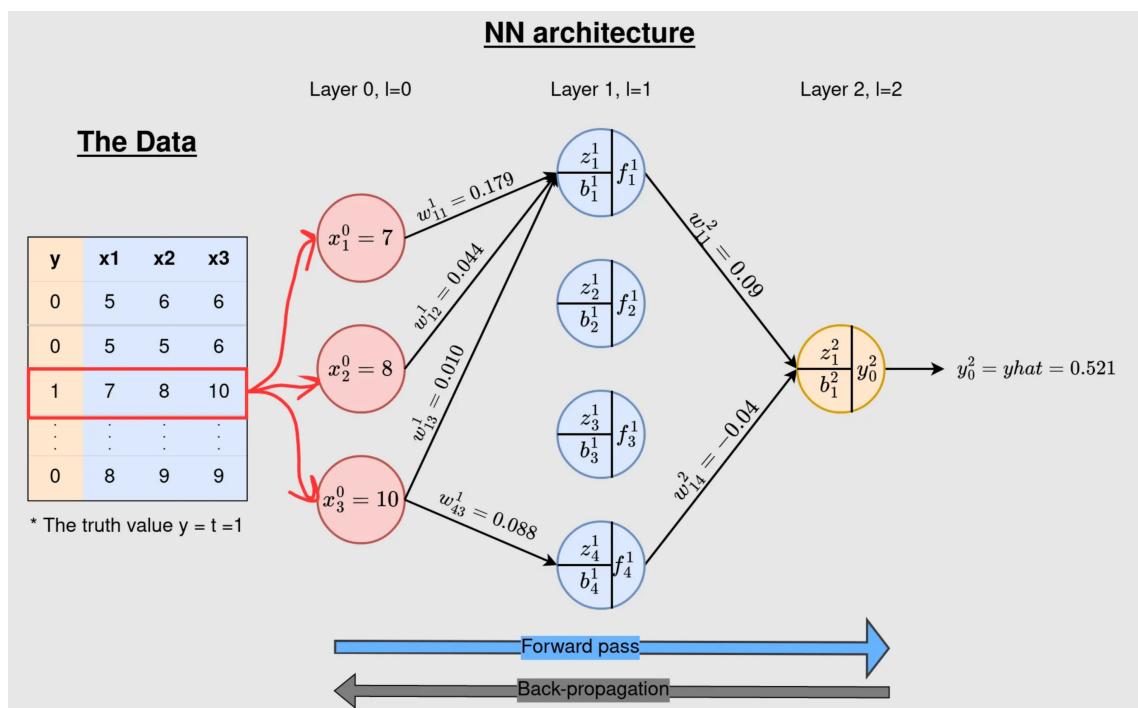


Figure 2.4: Example basic Neural Network Architecture Credits: Towards Data Science

Following we try to explain the simple weight optimization process with backpropagation:

1. Activation Function (Sigmoid): The sigmoid function, $\sigma(z)$, and its derivative are given by:

$$\begin{aligned}\sigma(z) &= \frac{1}{1 + e^{-z}} \\ \sigma'(z) &= \sigma(z)(1 - \sigma(z))\end{aligned}$$

2. Loss Function (Cross-Entropy Loss): For binary classification tasks, the cross-entropy loss for a predicted output y and true target t is:

$$L(y, t) = -t \log(y) - (1 - t) \log(1 - y)$$

3. Gradient of the Loss: The gradient of the cross-entropy loss with respect to the output y is:

$$\frac{\partial L}{\partial y} = \frac{y - t}{y(1 - y)}$$

4. Backpropagation: For a single-layer neural network, the gradient of the loss L with respect to the weight w is:

$$\begin{aligned}\frac{\partial L}{\partial w} &= \frac{\partial L}{\partial y} \times \frac{\partial y}{\partial z} \times \frac{\partial z}{\partial w} \\ \frac{\partial L}{\partial w} &= \frac{y - t}{y(1 - y)} \times \sigma(z)(1 - \sigma(z)) \times x\end{aligned}$$

where $z = w \cdot x + b$.

5. Weight Update using SGD: The weight is updated as:

$$w_{\text{new}} = w_{\text{old}} - \alpha \times \frac{\partial L}{\partial w}$$

Where α is the learning rate.

Batch Normalization

Batch Normalization is a technique that improves the training stability and convergence speed of neural networks. It normalizes the intermediate feature representations within a neural network batch-wise. This process helps mitigate the vanishing and exploding gradient problems, enabling smoother and faster convergence during training. Batch Normalization also acts as a regularizer, reducing the need for other regularization techniques like dropout.

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

Where μ is the mean and σ^2 is the variance.

Transfer Learning

Transfer Learning leverages knowledge gained from one task or dataset to improve performance on another related task or dataset. This approach is incredibly useful when labeled data is scarce for the target task. Pre-trained models, especially those trained on massive datasets like ImageNet, provide valuable initializations for the model's parameters. Fine-tuning a pre-trained model on a specific task allows the model to learn task-specific features while retaining the general knowledge it gained from the pre-training.

However, there can be disadvantages such as negative transfer where the pre-training may misguide the learning on the target task, or overfitting if the target task's dataset is too small.

Hyperparameter Tuning

Hyperparameter Tuning is the process of finding the optimal set of hyperparameters that yield the best performance for a given model and task. Hyperparameters, such as learning rate, batch size, and network architecture, significantly impact the model's training and performance. Efficiently tuning these hyperparameters can lead to faster convergence and better model generalization. Techniques like grid search, random search, and Bayesian optimization help navigate the high-dimensional space of hyperparameters to find optimal combinations.

Performance Metrics

Performance Metrics quantify how well a model is performing on specific tasks. They provide insights into the model's strengths and weaknesses.

For classification tasks:

- **Accuracy:** It is the ratio of correctly predicted instances to the total instances.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

- **Precision:** It is the ratio of correctly predicted positive observations to the total predicted positives.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- **Recall (Sensitivity):** It is the ratio of correctly predicted positive observations to all the actual positives.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- **F1-Score:** It is the weighted average of Precision and Recall. It tries to find the balance between precision and recall.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **AUC (Area Under The Curve) ROC (Receiver Operating Characteristics):** It measures the entire two-dimensional area underneath the entire ROC curve (from (0,0) to (1,1)). AUC represents the degree or measure of separability, indicating how much the model distinguishes between the positive and negative classes.

For regression tasks, Mean Absolute Error (MAE) and Mean Squared Error (MSE) measure the deviation between predicted and true values, giving a clear picture of the prediction quality.

- **Mean Absolute Error (MAE):**

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

- **Mean Squared Error (MSE):**

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Data Augmentation

Data Augmentation is essential for training robust and generalized models, particularly in domains like computer vision. By applying domain-specific transformations to the training data, the model becomes less sensitive to variations in the input data, such as rotation, scaling, and cropping. This technique effectively increases the diversity of the training dataset, helping the model learn more representative features and improving its ability to handle real-world data.

Learning Rate Schedulers

Learning rate schedulers are algorithms or methods that adjust the learning rate during training based on the number of epochs, iterations, or other criteria. The learning rate is a crucial hyperparameter in training deep neural networks. If set too high, the training might diverge; if set too low, the training might be too slow or get stuck in local minima. By dynamically adjusting the learning rate, schedulers aim to combine the benefits of both high and low learning rates, leading to faster convergence and better generalization.

Here are some popular learning rate schedulers:

1. **Step Decay:** Reduces the learning rate by a factor after a specified number of epochs.
2. **Exponential Decay:** Decreases the learning rate at each epoch or iteration exponentially.
3. **Cosine Annealing:** Adjusts the learning rate according to a cosine function.
4. **ReduceLROnPlateau:** Reduces the learning rate when a metric has stopped improving.
5. **Cyclic Learning Rates:** Allows the learning rate to cyclically vary between two bounds.

We focus on the exponential LR and Cosine Annealing LR schedulers.

The formula for exponential decay is:

$$\text{lr}(t) = \text{lr}_{\text{initial}} \times \text{factor}^t \quad (2.1)$$

Where:

- $\text{lr}_{\text{initial}} = 0.001$
- $\text{factor} = 0.5$

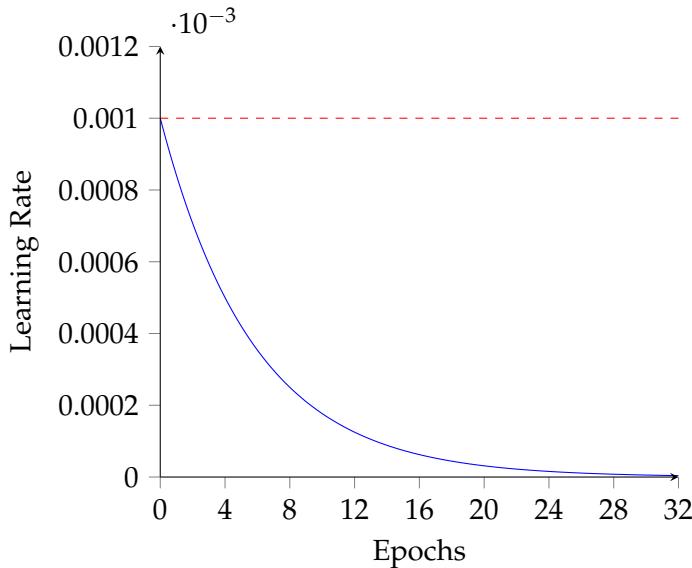


Figure 2.5: Exponential Learning Rate Schedule (Decay Factor = 0.5)

The formula for cosine annealing is:

$$\text{lr}(t) = \text{lr}_{\min} + \frac{1}{2}(\text{lr}_{\max} - \text{lr}_{\min})(1 + \cos(\frac{\pi t}{T})) \quad (2.2)$$

Where:

- $\text{lr}_{\min} = 0$ (usually)
- $\text{lr}_{\max} = 0.001$
- $T = 32$

In the figure ?? we can see the graph cosine annealing learning rate schedule.

Early Stopping

Early Stopping is a regularization technique used to prevent overfitting. Training a model for too many epochs can lead to it memorizing the training data rather than generalizing to new data. Early Stopping monitors the validation performance and halts training when the validation loss starts to increase. This prevents the model from deteriorating on unseen data and helps achieve the best trade-off between training performance and generalization.

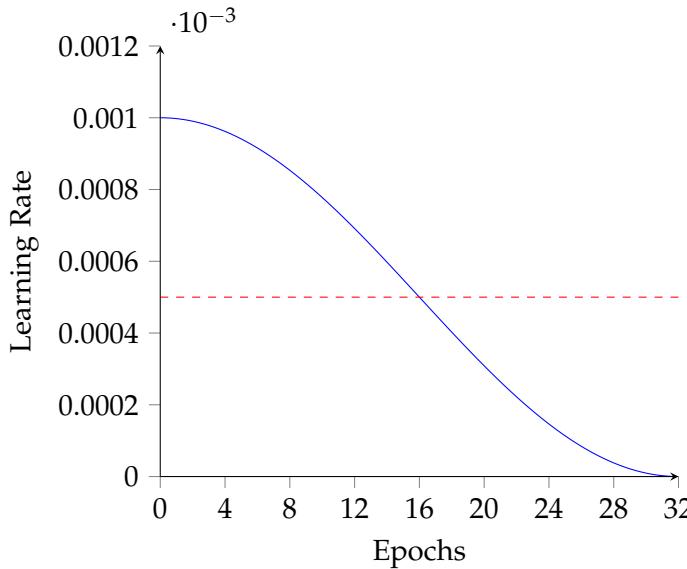


Figure 2.6: Cosine Annealing Learning Rate Schedule

2.1.4 Overview of Deep Learning Algorithms

Deep learning, over the years, has evolved to cater to diverse domains and challenges, leading to the development of several specialized algorithms. These algorithms emerged in response to specific challenges in data representation, computational efficiency, or domain-specific nuances. The diversity in algorithms is primarily a result of attempts to optimize performance across a myriad of tasks.

In chronological order, some of the influential deep learning architectures include Feedforward Neural Networks, Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, Gated Recurrent Units (GRUs), and Transformer Networks.

Algorithm	Description
Feedforward Neural Networks	Early architectures designed for pattern recognition without any cycles or loops.
CNNs	Specialized for processing grid-like data, such as images, using convolutional layers.
RNNs	Designed for sequential data, containing loops to maintain information across sequences.
LSTM	An RNN variant addressing vanishing gradient issues and retaining long-term dependencies.
GRUs	Simplified version of LSTMs, offering similar capabilities with fewer parameters.
Transformer Networks	Attention-based models providing parallel processing capabilities and superior performance in sequence tasks.

Table 2.2: Brief overview of key deep learning algorithms.

Convolutional Neural Networks

CNNs, emerging around the 1980s and popularized in the late 2000s, revolutionized image processing tasks. Unlike traditional Feedforward Neural Networks, CNNs employ convolutional layers, which use filters to scan an input for patterns, significantly reducing the number of parameters and enabling the model to recognize local patterns in data.

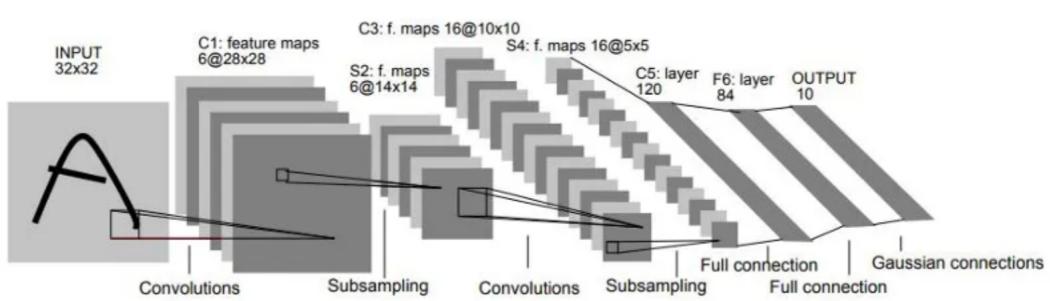


Figure 2.7: Example structure of a CNN [7].

Advantages of CNNs:

- *Parameter Efficiency*: Reduced parameters due to shared weights in convolutional layers.
- *Translation Invariance*: Ability to recognize patterns regardless of their position in the input.
- *Hierarchical Feature Learning*: Deep architectures extract layered features, moving from basic to complex.

Transformer Networks

Introduced in the "Attention Is All You Need" paper by Vaswani et al. in 2017, Transformers have since dominated various sequence-based tasks, especially in natural language processing. Instead of relying on recurrence, they use self-attention mechanisms to weigh the importance of different parts of the input data.

Advantages of Transformer Networks:

- *Parallel Processing*: Lack of recurrence allows simultaneous processing of sequence data, leading to speed gains.
- *Long-Distance Dependencies*: Captures relationships in data regardless of the distance between elements.
- *Scalability*: Easily scales to handle vast datasets and offers state-of-the-art results in many domains.

Applications:

- Natural Language Processing tasks like translation, summarization, and question-answering.
- Time-series forecasting.
- Some computer vision tasks leveraging the Vision Transformer architecture.

2.1.5 Applications of Deep Learning

Deep learning, an advanced subset of machine learning, has fostered a plethora of innovations across numerous domains due to its unparalleled proficiency in handling vast datasets and extracting intricate patterns. The applicability of deep learning transcends sectors, enabling tasks that were once considered the realm of science fiction.

- **Computer Vision:** From basic image classification to advanced tasks like object detection, segmentation, and facial recognition, deep learning, particularly through Convolutional Neural Networks (CNNs), has redefined the boundaries of what machines can perceive. Autonomous vehicles, medical image analysis, and augmented reality are just a few sectors harnessing the power of deep learning-driven computer vision.
- **Natural Language Processing (NLP):** Transformer architectures, most notably the BERT and GPT series, have drastically improved machines' ability to understand and generate human language. This has led to improvements in machine translation, sentiment analysis, and chatbots.
- **Speech Recognition:** Voice assistants like Siri, Alexa, and Google Assistant are a testament to the prowess of deep learning in understanding and synthesizing human speech, making voice-activated systems more accurate and ubiquitous.
- **Healthcare:** From diagnosing diseases with medical imaging to predicting patient trajectories, deep learning is assisting medical professionals by providing tools that can spot symptoms and patterns often too subtle for the human eye.
- **Finance:** In the world of finance, algorithms can predict stock market fluctuations, detect fraudulent activities, and automate trading by leveraging deep learning models.
- **Entertainment:** Deep learning-driven recommendation systems, such as those employed by Netflix and Spotify, personalize content suggestions, enhancing user experience. Also, Generative Adversarial Networks (GANs) have been used for art creation, game design, and even music generation.

While each of these domains has been transformed by the introduction of deep learning, our primary focus will be on **Computer Vision**. In the subsequent sections, we will delve deep into its intricacies, methodologies, and advancements, painting a comprehensive picture of how machines 'see' and 'interpret' visual data.

2.2 Computer Vision

Computer Vision (CV) is an interdisciplinary field that seeks to enable machines to interpret and make decisions based on visual data. Drawing inspirations from human vision, pattern recognition, and computational intelligence, CV has emerged as one of the most significant application areas for deep learning. As *Hubel and Wiesel* pointed out in their groundbreaking studies on the visual cortex [11], understanding vision is quintessential to understanding intelligence itself.

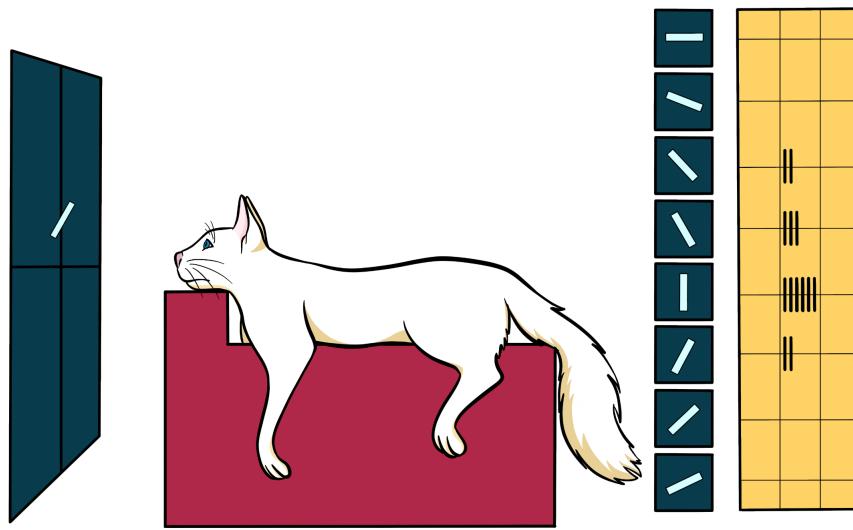


Figure 2.8: Simplified representation of Hubel and Wiesel's findings on the visual cortex of cats. Credits: FutureLearn

2.2.1 Image Classification

At its core, Image Classification is one of the fundamental tasks in computer vision. It involves assigning a predefined label to an input image, usually based on its primary content. For instance, an image containing predominantly a dog would be labeled "dog", irrespective of the breed or its position in the image. In mathematical terms, given an image I , a classifier function f assigns it a label l from a set of predetermined labels L :

$$l = f(I)$$

where $l \in L$.

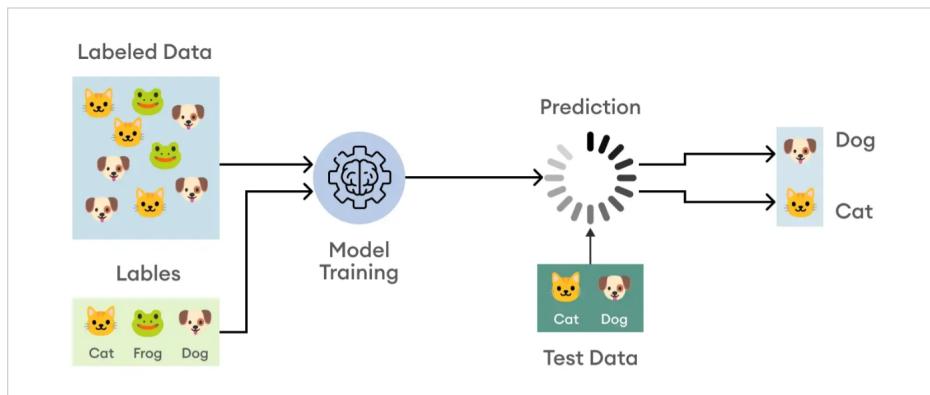


Figure 2.9: Simple illustration of image classification. Credits: SuperAnnotate

Applications of Image Classification: While the rudimentary idea behind image classification might seem simple, its applications have profound impacts on various sectors:

- *Medical Imaging*: Diagnosing diseases by classifying medical images into categories like 'tumor' or 'no tumor'.
- *Structural Health Monitoring*: Detecting defects or damages in structures like bridges, buildings, and dams by processing and analyzing images or videos.
- *Agriculture*: Identifying unhealthy plants or predicting the type of crops in satellite imagery.
- *Security*: Automated surveillance systems detecting unauthorized or suspicious individuals.
- *Retail*: Assisting in automated checkout processes by identifying products.
- *Automotive*: In autonomous driving, classifying objects helps vehicles make informed decisions, e.g., distinguishing between pedestrians and lampposts.
- *Smart Cities*: Analyzing satellite or drone imagery to plan urban development, detect changes in land use, or estimate population densities.
- *Finance*: Document classification to identify types of financial statements or bills.
- *Social Media*: Content recommendation based on user's photo uploads or identifying inappropriate content.

Challenges in Image Classification:

Despite advances, several challenges remain in image classification:

- *Intra-class Variation*: Objects of the same class can appear different under varying lighting, angles, or occlusions.
- *Scalability*: As the number of categories increases, distinguishing between them becomes harder.
- *Data Imbalance*: Some classes might have fewer training samples than others, leading to biased predictions.
- *Transferability*: A model trained on one dataset might not perform well on another due to domain shifts.

2.3 Reproducibility in Scientific Research

Reproducibility stands as the foundational pillar upon which the foundation of the scientific method has been built. It functions as a strict check, ensuring that scientific findings remain steady and consistent, regardless of who conducts the experiment.

In research, methodological variability often appears as a major challenge. The slightest changes in methodology or experimental procedure can significantly change outcomes. For instance, an experiment conducted at a slightly different temperature or using a slightly different concentration of a reagent can produce different results. As we delve further into research, especially in areas overflowing with big data and high-throughput technologies, the challenges increase. Managing and processing such large amounts of data to ensure reproducibility is a challenging task.

Today's research isn't just about test tubes and microscopes; it's closely tied with software. However, as software changes and updates, it introduces another source of inconsistency. An older experiment rerun on a newer software version might yield different results, making reproducibility unclear. This reliance on software, along with the bias towards publishing positive or novel results, means that a significant portion of research, especially those with negative results, remains unpublished.

Furthermore, the complexity of the reproducibility issue is highlighted by a survey conducted by Nature, where more than 70% of the surveyed researchers admitted to having faced challenges in reproducing another scientist's experiments, and over half encountered difficulties reproducing their own experiments. Despite this, most still trust the published literature, showing mixed feelings about reproducibility in the scientific community [12].

However, the consequences of irreproducible research extend beyond the academic world. The impact of irreproducible research is felt throughout society, leading to wasted resources as other researchers unknowingly follow paths based on incorrect findings. Even more serious, in fields like medicine, the risks are high. Irreproducible research can lead to misguided clinical practices, resulting in wrong treatment methods that can harm patients.

2.3.1 Irreproducibility in Deep Learning

Deep within the neural networks of deep learning lies an inherent stochastic nature that brings with it both challenges and opportunities. This randomness, while fundamental to the training processes of deep learning, also introduces a level of unpredictability that can be both a boon and a bane.

At the heart of this randomness is the initialization of weights in neural networks. Starting a model's training journey, these weights are often set to random values, acting as the initial step in a long journey towards optimization. Yet, like setting out on a hike from different starting points, these varied initializations can lead the model to different local optima, influencing the final model's performance.

Data augmentation, a technique employed to artificially expand training datasets, further introduces variability. By applying transformations like random cropping or rotation, each epoch of training might expose the model to subtle variations of the same data. While this enhances model robustness, it's another source of randomness.

Yet, the landscape of randomness in deep learning isn't just about data and weights. Algorithms like Stochastic Gradient Descent (SGD) introduce their own flavor of unpredictability. By using a random subset of data for weight updates, SGD ensures that the model doesn't just memorize data but learns the underlying patterns. However, this very strength is also a source of variability.

Beyond these, computational intricacies, such as floating-point precision in digital computations, especially on GPUs, add their own minute variations. Over millions of operations, these tiny discrepancies accumulate, causing significant variability in outcomes.

The randomness issue in deep learning is being actively researched. Several studies have approached the issue from various angles. For instance, Gundersen et al. [13], identified the sources of irreproducibility. They found that non-technical sources, such as the initial conditions and the environment of the experiments, affect reproducibility. These causes are often related to the mindset and the thinking of the researchers as well as financial constraints. Fig ?? shows the non-technical factors that make harder to reproduce any scientific results. These factors indeed valid for deep learning tasks. Detailed *documentation* with instructions to reproduce to results will help with the reproducibility and *transparency* on this process would certainly contribute to this cause. Lastly, some *ethics and privacy* regulations could also prevent researchers to reveal sensitive information, especially in medical domain.

We examine the randomness in this master thesis, which is a technical factor. There are implementation factors and algorithmic factors. These are generally related with the tooling and the methodology of the experiments. These factors introduce randomness on different level of deep learning process as a result, variance in the performance occurs. According to Pham et al. [14], accuracy of the models varies up to 10.8%. They conducted a survey as well and find out that 83.8% of the participants unaware of or unsure about implementation level variance. Thus, it is reasonable to approach implementation level randomness interrogatively and investigate some standard processes in DL.

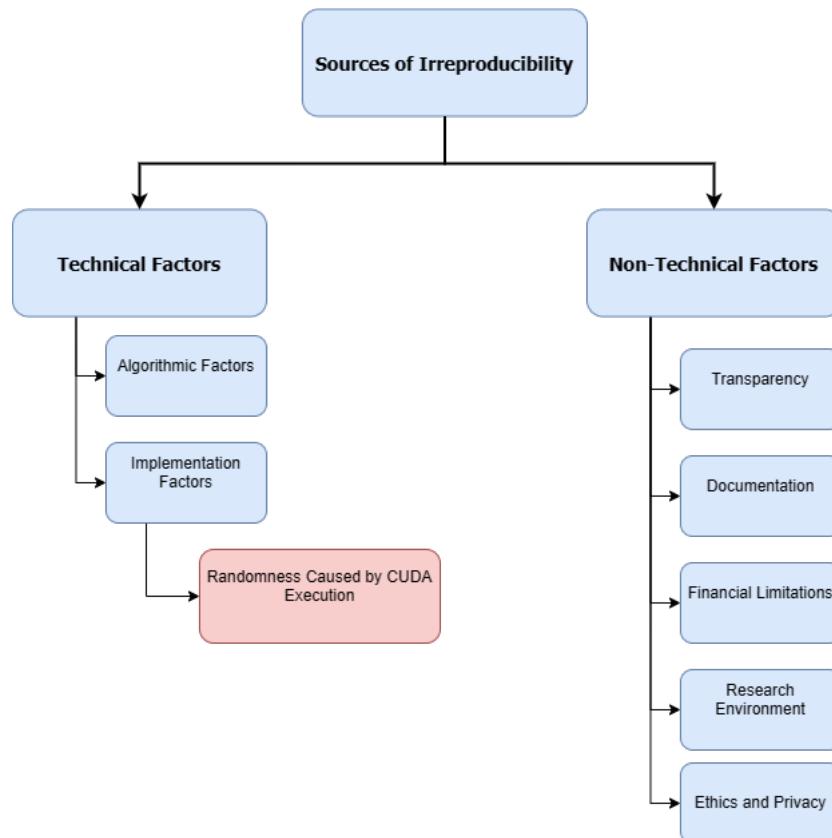


Figure 2.10: Investigation area of the study.

The ripples of this randomness in deep learning extend far and wide. As AI systems become ever more pervasive, from diagnosing diseases to driving cars, unpredictabil-

ity can pose substantial challenges. Models that yield varying results across runs can complicate evaluations, making direct comparisons challenging. In high-stakes scenarios, like medical diagnoses, this unpredictability can have dire ramifications. Hence, while randomness is an intrinsic aspect of deep learning, understanding, managing, and sometimes mitigating it becomes paramount to harness the true potential of AI.

2.3.2 Sources of Randomness

Table ?? presents a full list of algorithmic and implementation factors that introduces randomness [14] [13] [2]:

Algorithmic Factors	Implementation Factors
Nondeterministic DL layers that introduce stochasticity	Used framework
Random initialization of the weights	Used framework version
Hyperparameter optimization	Nondeterministic floating point arithmetic
Data augmentation	Parallel execution
Data shuffling and ordering	Auto selection of primitive operations
Batch ordering	Unpredictability of processing unit

Table 2.3: Algorithmic and Implementation Randomness Factors in Deep Learning.

2.3.3 Floating-Point Arithmetics and Parallel Execution

In the high-performance computing, GPUs have emerged as game-changers. With their vast array of parallel processing units, they have brought unprecedented computational capabilities to the fingertips of developers. NVIDIA's CUDA platform harnesses this power, providing a framework for parallel computing on CUDA-capable GPUs [15]. At the heart of this acceleration lies the intricate dance between floating-point arithmetic and parallel execution.

Floating-Point Arithmetic: A Double-Edged Sword

Floating-point arithmetic provides a method to represent and perform operations on real numbers using a finite number of bits. This mathematical framework is fundamental for a multitude of scientific calculations, including those prevalent in deep learning (DL) networks. The primary challenge is that the finite precision of floating-point arithmetic can lead to errors. Although these errors are generally minuscule, their effects can accumulate, notably in iterative algorithms.

The IEEE 754 standard delineates the specifics of floating-point arithmetic, fostering consistency across various platforms. This consistency is indispensable, especially given

the increasing use of computational methods in scientific research. However, one limitation of the IEEE 754 standard is that, while it prescribes the precision and behavior of singular operations, it doesn't fully govern the sequence of operations or their compound effects. This issue becomes particularly pronounced in parallel computing environments. Here, operations can run concurrently or in an unpredictable sequence, leading to the propagation and magnification of these inherent errors, sometimes resulting in non-deterministic outcomes.

Understanding the Limitations:

1. *Representation Limits:* Not all real numbers can be precisely represented due to the fixed number of bits allocated for floating-point numbers. For instance, the fraction $\frac{1}{3}$ is a recurring decimal, and its binary representation is similarly recurring. As a result, it's truncated, introducing an error.
2. *Rounding Errors:* Numbers that can't be accurately represented lead to rounding errors during operations. For a practical demonstration, in an ideal scenario, $0.1 + 0.2 = 0.3$. Yet, in floating-point arithmetic, this sum might manifest as 0.3000000000000004 due to inherent representation errors.
3. *Error Accumulation:* The cumulative effect of small errors can be significant in iterative processes. Consider an algorithm that repeatedly adds a minuscule value. If the algorithm adds 0.0000001 a million times, the expected result is 0.1 . However, in floating-point arithmetic, the final sum might diverge slightly from this value.
4. *Catastrophic Cancellation:* Subtraction between nearly equivalent floating-point numbers can obliterate significant digits. As an illustrative example, for numbers $a = 1.0000001$ and $b = 1.0000002$, the difference $b - a$ should be 0.0000001 . If the precision limits of the system coerce both numbers to round to 1.000000 , the resultant difference becomes zero—a stark contrast to the true value.

In summary, while floating-point arithmetic is a cornerstone of computational methodologies, particularly in DL, its limitations necessitate meticulousness and often compensatory strategies in algorithm design and execution.

Parallel Execution in CUDA

CUDA's parallel execution model divides tasks into threads that are executed across the multiple cores of a GPU. These threads are grouped into blocks, and these blocks are, in turn, organized into grids [15]. This hierarchy allows CUDA to scale computations across different GPU architectures effectively. The figure below illustrates this structure:

However, the inherent nature of parallel execution introduces randomness due to the unpredictable order in which threads are completed. This randomness is exacerbated when combined with the nuances of floating-point arithmetic. Two runs of the same parallel operation can produce slightly different results due to the varying order of execution and the consequent accumulation of floating-point errors.

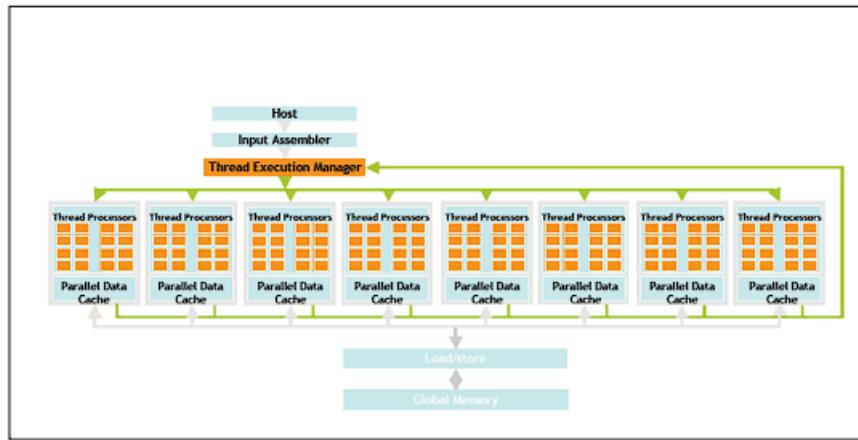


Figure 2.11: Architecture of CUDA-capable GPU. Source: [15]

cuDNN: Leveraging CUDA and Introducing Nondeterminism

cuDNN is a deep neural network GPU-accelerated library that builds upon CUDA [15]. While it greatly boosts the efficiency of DL networks, some of its operations, such as the CUDA convolution benchmarking, are nondeterministic. This nondeterminism stems from both the scheduling of parallel tasks and the intricacies of floating-point precision. Deterministic execution schemes have been proposed to counteract this issue [16]. However, these schemes can potentially introduce computational overhead.

The Trade-Off: Performance vs. Reproducibility

There's a delicate balance between performance and reproducibility in parallel computing. Deterministic execution schemes increase the reproducibility and credibility of DL networks, making results more trustworthy and comparable across runs. However, they may come at the cost of increased computational time, which could be a significant drawback in time-sensitive applications or large-scale computations. It is, therefore, essential to analyze and understand these trade-offs carefully, tailoring solutions to specific use-cases and experimental setups.

In conclusion, the interplay between floating-point arithmetic and parallel execution in CUDA is a complex one. While it offers immense computational power, it also brings forth challenges in reproducibility. As researchers and developers, it's our responsibility to approach these challenges head-on, understanding their roots and implications, and crafting solutions that balance both performance and reproducibility.

2.4 Frameworks and Tools

The success and efficiency of any scientific study often hinge on the careful selection and proficient use of the right computational tools and frameworks. In the scope of this study, several tools and frameworks have been adopted, each with its unique benefits and contributions to the overall experiment. Here, we provide a brief overview of these tools and their significance in the research.

The synergy of these tools and frameworks enabled a seamless, efficient, and insightful experimental process. PyTorch offered the foundational deep learning capabilities; Weights and Biases ensured that experiments were tracked, visualized, and optimized;

Framework/Tool	PyTorch
Publisher	Facebook's AI Research lab (FAIR)
Overview	An open-source deep learning framework with a dynamic computational graph, PyTorch is versatile and suited for research. It allows on-the-fly graph modifications.
Role in the Experiment	PyTorch was central to our study, aiding in the design, training, and evaluation of deep learning models. Its adaptability supported quick prototyping and its extensive library eased the deployment of various architectures and methodologies.

Table 2.4: Overview of PyTorch

Framework/Tool	Weights and Biases (W&B)
Publisher	Weights & Biases, Inc.
Overview	W&B is crafted to assist researchers in tracking and visualizing ML experiments. It encapsulates features like hyperparameter tuning, model visualization, and performance tracking.
Role in the Experiment	W&B was integral for experiment management, used for logging results, visualizing model dynamics, and comparing architectures and hyperparameters.

Table 2.5: Overview of Weights and Biases

and SLURM managed the computational resources effectively. Together, they ensured that the research was conducted in a structured, efficient, and reproducible manner.

2.5 Mathematical Evaluation

In the thesis, several mathematical tools and methods are used to evaluate the results. In this section, we explain these tools and methods.

Standard Variance

Standard Variance, denoted as σ^2 , is calculated using the formula:

$$\sigma^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

where:

- n is the number of observations,
- x_i is each individual observation,
- \bar{x} is the mean of all observations.

Framework/Tool	SLURM
Publisher	SchedMD
Overview	SLURM is an open-source job scheduler, pivotal for resource allocation in multi-user clusters. It's a mainstay in high-performance computing due to its scalability.
Role in the Experiment	Given the computational demands of deep learning, SLURM managed our job submissions. It optimized resource allocation, enabling parallel experiment execution without contention.

Table 2.6: Overview of SLURM

Standard Variance is crucial in machine learning and statistics to understand the dispersion and to normalize the data. It's also vital for gauging the accuracy and the reproducibility of models by analyzing the variance in predictions.

Cosine Similarity

Cosine Similarity is a metric used to measure how similar two vectors are, regardless of their size. The formula for cosine similarity S_C between two vectors A and B is given as:

$$S_C(A, B) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

where n is the dimension of the vectors. Cosine similarity is extensively used in machine learning for understanding the similarity between vectors or entities, which is fundamental in algorithms like nearest neighbors, clustering, and also in recommendation systems.

T-Test with Population Mean

The One Sample T-Test is a statistical method used to determine whether the mean of a single sample is significantly different from a known or hypothesized population mean. The formula for the one-sample t-test is given by:

$$t = \frac{(\bar{x} - \mu)}{(s / \sqrt{n})}$$

where:

- \bar{x} is the sample mean,
- μ is the theoretical population mean,
- s is the sample standard deviation,
- n is the sample size.

The population mean can also be estimated as:

$$\text{Population mean} = \text{Sample mean} + T \times (\text{Standard error of the mean})$$

Kruskal-Wallis Test

The Kruskal-Wallis test is a non-parametric method for testing whether samples originate from the same distribution. It is used for comparing more than two samples that are independent, or not related. The formula for the Kruskal-Wallis test is:

$$H = (N - 1) \frac{\sum_{i=1}^g n_i (\bar{r}_{i\cdot} - \bar{r})^2}{\sum_{i=1}^g \sum_{j=1}^{n_i} (r_{ij} - \bar{r})^2}$$

where:

- N is the total number of observations across all groups,
- g is the number of groups,
- n_i is the number of observations in group i ,
- $\bar{r}_{i\cdot}$ is the average rank of group i ,
- \bar{r} is the average rank of all the data,
- r_{ij} is the rank of observation j in group i .

ANOVA Test

Analysis of Variance (ANOVA) is a statistical method used to test differences between two or more means. It divides the variance in a dataset into component parts associated with different sources of variation. In its simplest form (One-Way ANOVA), it can be used to compare the means of three or more independent groups. ANOVA partitions the total sum of squares into components related to the effects used in the model, as expressed in:

$$SS_{\text{Total}} = SS_{\text{Error}} + SS_{\text{Treatments}}$$

The F-test statistic in ANOVA is calculated as:

$$F = \frac{MS_{\text{Treatments}}}{MS_{\text{Error}}} = \frac{SS_{\text{Treatments}}/(I - 1)}{SS_{\text{Error}}/(n_T - I)}$$

where:

- MS is the mean square,
- I is the number of treatments,
- n_T is the total number of cases.

3 Literature Review

This empirical study examines reproducibility and randomness in the AI field. In our review of existing literature, we found limited work that allows for a direct comparison with our results. Nevertheless, a number of relevant studies have addressed aspects of these topics, offering insights and methodologies that are pertinent to our investigation. Below, we provide a concise overview of these works, setting the stage for our own contributions.

Goodman et al. [3] defined the term reproducibility that we use in this study. Raste et al. [17] conducted an empirical study to investigate the impact of Randomness in several machine learning algorithms. They concluded that transparency in used methods and datasets is a crucial part to create reproducible results. Chen et al. [5] listed the challenges to reproduce any result of a deep learning model. They emphasized the importance of the reproducible DL models. They also introduced their approach and solution to mitigate the challenges. Furthermore, they supported their approach with case studies and presented guidelines. Scarpadane et al. [18] gave an overview of randomness and how they applied in deep learning research. Dirnagl et al. [19] introduced a different perspective on research reproducibility in different research areas. It can be understood that in the past years, researchers have been investigating reproducibility and randomness from different perspectives. There are guidelines on how to achieve reproducibility. Some studies emphasize the importance of the problem. If the aim is to enhance the transparency, accountability and trustworthiness of AI models, then research on Reproducibility in AI research area could fall under XAI as a topic.

However, reproducibility in the AI research area could also be considered as a separate topic from XAI if the goal is to improve the methods and practices of conducting and reporting AI experiments. For example, Pranava et al. [20] stated that deep neural network-based models are sometimes vulnerable to randomness during the training of the models. They investigated the random-seed-based perturbations and proposed a solution to mitigate standard deviations of the model performance. Pham et al. [14] analyzed the variance in deep learning software systems. With the immense amount of GPU time, they use widely-used datasets and models with core DL libraries to perform experiments. Their core contributions are the variance in performance that can be up to 10.8% and a survey which results suggested that a high percentage of researchers in the area are not aware of the variances. Another valuable empirical study is made by Summers et al. [21]. Their core contribution is that the nondeterminism factors result in similar levels of variability. Shamir et al. [22] proposed an anti-distillation to improve reproducibility in deep networks. Snapp et al. [23] conducted another empirical study with simple models to analyze the irreproducibility in deep networks. They found out that even with the simplest model reproducibility can be a challenge. Shallue et al. [24] demonstrated the properties of the batch size number and its relation with out-of-sample error. They found that max useful batch size depends on the properties of the model, training algorithm and dataset. Fellicious et al. [25] investigated the different optimizers and architectures with respect to varying initial weights. As can be seen, there are

numerous empirical studies conducted by researchers in recent years. Their findings and methods to limit or control randomness will be used in this thesis to test the use cases. We plan to use deterministic execution methods from the literature. It is not our goal to propose a new method.

A notable contribution to the discourse on reproducibility and the role of randomness was made by Zhuang et al. [2], who established a benchmark setup to delineate the attributes of tooling in managing randomness. They posited a binary stance on nondeterminism, advocating for its comprehensive control, failing which, they argue, any control exerted becomes moot. Their investigation further revealed a differential sensitivity to randomness across various subsets of datasets. Moreover, they highlighted the substantial overhead incurred by deterministic approaches. An in-depth exploration into hardware and CUDA-induced randomness was undertaken, wherein the sources of randomness were categorized and analyzed in groups relative to different architectures. The distinct factors of implementation and algorithm were individually regulated and juxtaposed against varying architectures, providing a nuanced understanding of the interplay between these elements and randomness. In a parallel vein, Chou et al. [16] presented a proposition for deterministic execution on GPU platforms, identifying it as a facilitator of reproducibility. This feature will be harnessed in this study to attain deterministic executions whenever there's a necessity to regulate CUDA-induced randomness. Such deterministic controls are instrumental in isolating and understanding the variations in the outcomes, thereby enhancing the reproducibility and interpretability of the results.

In the medical sector, reproducibility is a significant concern, especially in Machine Learning for Health (MLH). McDermott et al. [26] conducted a study to understand the reproducibility challenges in MLH, proposing a taxonomy and defining the broad goals and challenges surrounding reproducibility in this domain. Likewise, Beam et al. [27] outlined the specific challenges faced when attempting to reproduce machine learning models in healthcare. In this thesis, one of the use cases is derived from the medical domain. Although a public dataset is employed for this use case, many tasks in the medical domain are hindered by the lack of publicly available datasets. The accessibility of data is a common issue as, oftentimes, datasets in the medical domain are private due to the sensitive nature of medical data. To investigate such tasks from a reproducibility standpoint, special permissions are required to access the necessary data. The unavailability of public datasets not only hampers the reproducibility and validation of machine learning models but also inhibits collaborative research efforts within the community.

4 Research Methodology

To investigate the issue of irreproducibility empirically, we built an experimental setup and a working pipeline. Consistency across runs should be maintained as much as possible by using the same settings and established methods for control. By accounting for other sources of randomness, we aim to isolate and document the impact of CUDA execution-related randomness. This will involve analyzing performance variance for patterns and correlations. The credibility of the results will be ensured by the correct isolation of the randomness. We execute the multiple runs while carefully managing all sources of randomness. The exact number of runs depends on factors such as GPU capabilities and environmental conditions, but we anticipated 180 independent runs overall. The reasoning for this number will be explained later in this chapter.

4.1 Experiment Design and Objectives

Our investigation revolves around the question of the influence of CUDA-induced randomness on the reproducibility and robustness of deep learning applications, specifically within the realm of computer vision. We have termed this central investigation as the Main Question (MQ):

MQ: *What is the overarching impact of CUDA randomness on the reproducibility and performance of deep learning tasks in computer vision?*

To dissect this question and gain a more granular understanding, we have broken it down into three specific sub-questions:

1. Our first sub-question addresses the performance variation in deep learning tasks when we control for other potential sources of randomness but deliberately allow the inherent randomness from CUDA execution. The aim here is to gauge the degree to which CUDA randomness alone can influence the outcome. To this end, we execute five identical runs in each configuration to record the variance and ensure the reliability of our findings. This is encapsulated in:

SQ1: *What is the extent of performance variability when controlling for other sources of randomness, while allowing for randomness from CUDA execution to be present?*

2. The second sub-question explores the implications of using deterministic settings within CUDA. While deterministic approaches may provide reproducibility, they might come with their own set of trade-offs. By having deterministic CUDA execution in some configurations, we intend to shed light on any potential performance or computational costs associated with such an approach. This is summarized in:

SQ2: *What is the cost of using deterministic approaches in CUDA randomness?*

3. Lastly, our third sub-question addresses the practical implications of CUDA randomness in specific real-world domains. By utilizing two domain-specific datasets

— one from Civil Engineering and the other from Medicine — we aim to unearth the nuances of how CUDA randomness might differentially affect performance and computation costs in these specialized areas. This practical aspect of our research is framed as:

SQ3: *How does the randomness in Computer Vision impact the task performance and computation cost for specific applications, such as Civil Engineering and Medicine?*

By investigating these sub-questions, we hope to provide a comprehensive answer to our main research query, bridging the gap between theoretical understanding and practical implications of CUDA randomness in computer vision-based deep learning tasks.

One main solution to reduce the randomness in deep learning applications is fixing the seeds. Seeding is a fundamental practice in deep learning that allows the random number generator to produce same random numbers everytime. As mentioned, inherent randomness arises from, for example, initialization of weights, data shuffling, and data augmentation. This randomness can lead to variations in model performance and outcomes, even when the model is trained with the exact same parameters and data. By fixing the seed value, often referred to as "setting the seed", we can ensure up to some point that these random operations produce consistent results every time the model is run. This deterministic behavior is crucial when we wish to reproduce results, compare the efficacy of different models, or ensure consistent behavior across runs. For experiments, we think that different seed configurations might show different results and sensitivity to CUDA-caused randomness. Thus, we have five different seed configurations and these are chosen randomly.

The strategy detailed above can be better understood through the experimental structure illustrated in the figure ??.

As can be seen from the figure, there are *two* options available for randomness settings in CUDA: nondeterministic and deterministic. There are *five* separate runs using fixed seed configurations with nondeterministic CUDA settings while controlling all other sources of randomness. That means in the experiments, framework, used libraries and versions, hardware and other environmental factors will be kept the same. Additionally, we have *one* run using deterministic CUDA settings. Ultimetaly, We use *two* commonly used optimizers to find out the optimizer influence and conduct the tests on *three* different datasets, resulting in a total of $6*5*2*3 = 180$ runs.

4.2 Datasets

In this work, we use only publicly available sources. As a start, CIFAR-10 [28] dataset is used. Experiments on CIFAR-10 will provide neutral analysis that would represent the image classification tasks generally and help build the experimentation setup. Additionally, two different datasets from two different domains will be taken into account. Investigating these two use cases will unlock the opportunity to answer the fourth question. We use SDNET2018 [29] as the use case from Civil Engineering. There are several papers that for comparisons and validations. One example is the work of Dorafshan et

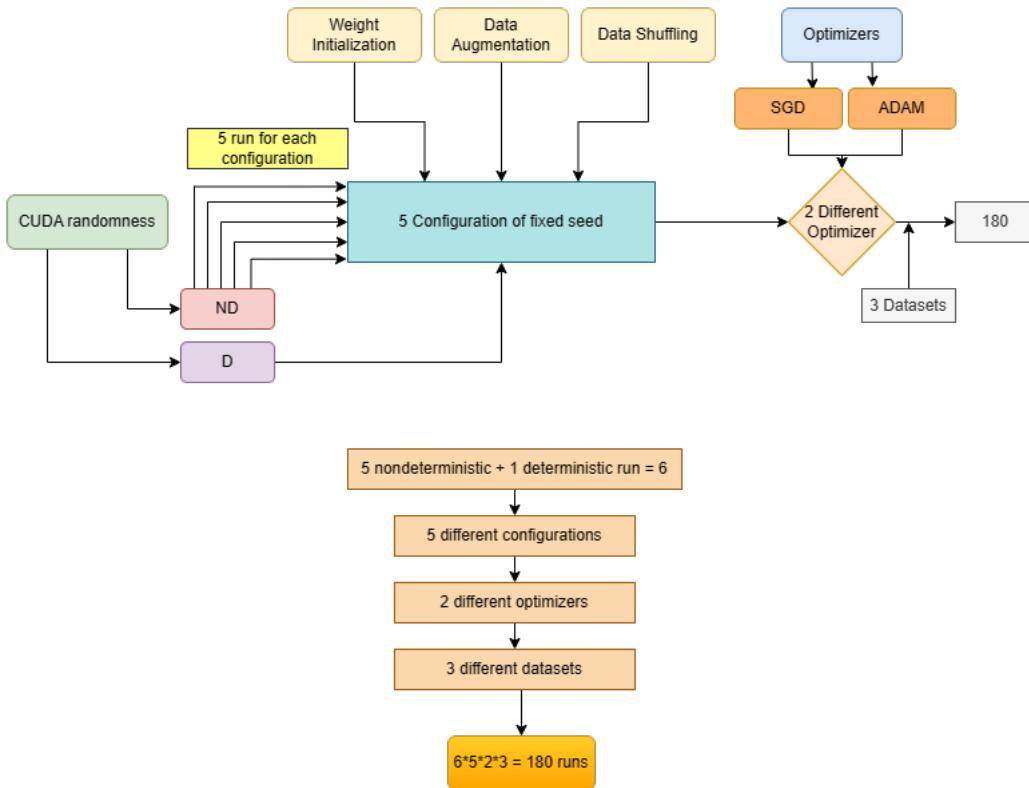


Figure 4.1: Structure of the experimentation

al. [30]. The researchers showcased their crack detection algorithm's performance by conducting a series of tests using the aforementioned dataset. The algorithm was developed based on the AlexNet Deep Convolutional Neural Network (DCNN) architecture [31], which enabled it to achieve accurate results. The dataset is publicly available and usable. In the Medical domain, the well-known breast cancer dataset CBIS-DDSM [32] is used. The choices of the datasets are made based on some criterias such as ease of applicability, availability, ease of implementation and the number of papers that use the dataset.

4.2.1 Overview of the Datasets

The CIFAR-10 dataset comprises 60,000 32x32 colour images categorized into 10 distinct classes. Each class encapsulates 6,000 images, aggregating to 50,000 training images and 10,000 testing images. The dataset is partitioned into five training sets and a singular testing set. Each training set contains 10,000 images, while the test set encompasses exactly 1,000 randomly chosen images from every category. The training sets exhibit a random order of images, with some sets potentially harboring a higher count of images from a particular category than others. However, cumulatively, the training sets possess an equal distribution with 5,000 images from each category. Table ?? delineates the labels within the dataset.

Figure ?? exhibits a selection of images from each class of the CIFAR-10 dataset, generated using a Python script. The figure elucidates the variability and characteristics inherent in each class, which is instrumental in understanding the complexity of the dataset. It's conspicuous that the dataset encapsulates a diverse range of features and

Labels	Type
0	Airplane
1	Automobile
2	Bird
3	Cat
4	Deer
5	Dog
6	Frog
7	Horse
8	Ship
9	Truck

Table 4.1: Labels in the CIFAR-10 Dataset

attributes within each class, which underscores the necessity for robust machine learning models capable of discerning nuanced differences and similarities among the images.

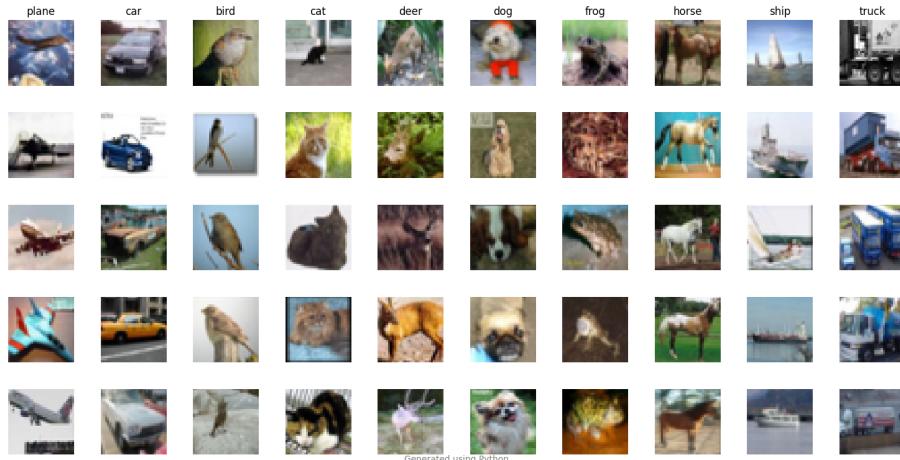


Figure 4.2: Example images from each class of the CIFAR-10 dataset.

SDNET2018 is a meticulously curated dataset of labelled images conceived to facilitate the training, testing, and evaluation of artificial intelligence algorithms aimed at detecting cracks in concrete structures. The dataset amasses over 56,000 images showcasing various concrete structures such as walls, pavements, and bridge decks, both with and without the manifestation of cracks. The spectrum of crack widths within the dataset spans from a minute 0.06mm to a noticeable 25mm. Additionally, the dataset encapsulates images exhibiting a myriad of obstructions like shadows, surface roughness, scaling, edges, holes, and background debris which introduce real-world complexity to the task of crack detection.

The primary objective is to harness machine learning and computer vision algorithms to accurately identify and localize cracks amidst a variety of structural backgrounds and obstructions. Such an endeavor is crucial for ensuring the structural integrity and safety of urban infrastructure, thereby contributing to public safety and cost-effective maintenance.

Figure ?? presents a selection of images from the SDNET2018 training set, generated using a Python script. The figure underscores the diverse scenarios under which the algorithms are expected to perform. The labeled images exhibit a range of crack manifes-

tations and obstructions, showcasing the real-world challenges inherent in automated crack detection. The variability in crack width, orientation, and background interference necessitates the development of robust algorithms capable of discerning subtle structural anomalies amidst diverse and cluttered scenes.

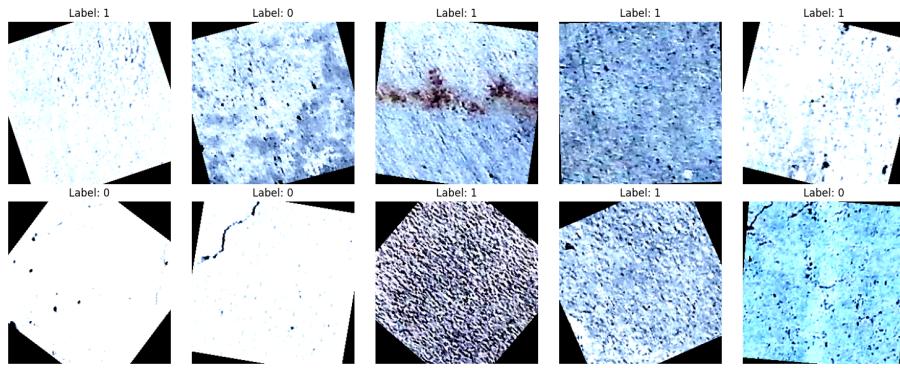


Figure 4.3: Random example images from the SDNET2018 training set with labels.

The CBIS-DDSM (Curated Breast Imaging Subset of DDSM) is a distinguished dataset in the domain of digital mammography research. It originates from the Digital Database for Screening Mammography (DDSM) and is curated to be more accessible for contemporary machine learning and computer vision applications.

The core of CBIS-DDSM is composed of mammographic images, encompassing both benign and malignant cases. The images are further categorized based on the type of lesion visible, such as masses or calcifications. The images in CBIS-DDSM have been converted to a more standard format to facilitate modern research, featuring improved lesion annotations and consistent metadata. Each image is accompanied by associated metadata, which may include the patient's age, the type of lesion, its location, and other relevant clinical details.

Due to its comprehensive nature and the diversity of cases it presents, the CBIS-DDSM has become an invaluable resource for researchers aiming to develop and validate breast cancer detection algorithms, especially those leveraging machine learning and computer vision.

The composition of the CBIS-DDSM dataset is detailed in Table ??, illustrating the distribution of images across different categories. The table segregates the images into benign and malignant cases, and further breaks down the data based on the type of lesions, i.e., masses and calcifications. It is evident from the table that the dataset provides a balanced representation of both benign and malignant cases, which is crucial for training machine learning models to recognize and differentiate between various types of lesions. Moreover, the distinct categorization of lesions into masses and calcifications aids in understanding the diverse nature of mammographic abnormalities present in the dataset, thereby enriching the scope of research in breast cancer detection.

Table 4.2: Composition of CBIS-DDSM Dataset

Category	Number of Images
Benign	1429
Malignant	1457
Masses	891
Calcifications	735

Figure ?? displays random example raw images from the CBIS-DDSM dataset, generated using a Python script. The images showcased serve as a stark reminder of the challenges inherent in mammographic image analysis. At a glance, it is difficult, especially for the untrained eye, to ascertain the precise location of cancerous lesions. The subtlety of mammographic abnormalities and the often-ambiguous nature of malignant versus benign indicators render the task of accurate identification and localization exceedingly challenging without substantial domain knowledge and expertise. Moreover, the visual similarity between benign and malignant lesions can further confound accurate classification, underscoring the necessity for sophisticated machine learning and computer vision algorithms capable of teasing apart these nuanced differences to aid in early and accurate breast cancer detection.

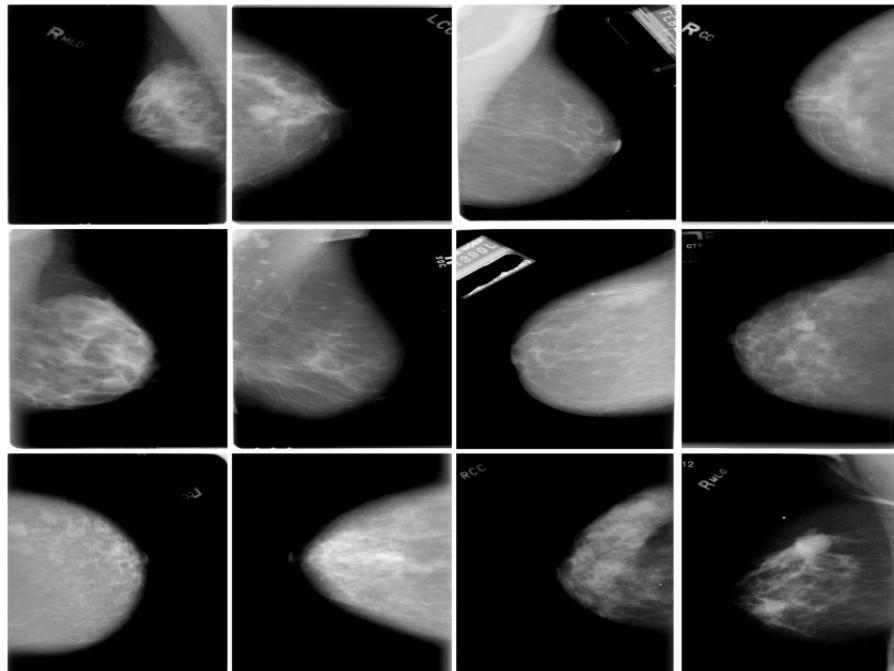


Figure 4.4: Random example raw images from the CBIS-DDSM dataset.

4.3 Model Architectures

The three tasks at hand involve classifying images and are of particular importance in the fields of civil engineering and medicine, where they have real-world applications and deal with critical domains. In order to accurately represent real-world scenarios, there is a strong inclination to achieve high performance on these tasks, which necessitates the use of recent and deep CNN [7] models. An example of a CNN structure is shown in Fig ???. CNNs extract features from images through convolutional and pooling layers. Convolutional layers apply filters to small regions of the image, producing feature maps that highlight patterns. Pooling layers downsample the feature maps to reduce their dimensionality. This hierarchical feature extraction enables CNNs to learn complex representations of the image, which can be used to classify the image into different categories. It is important to use CNN models that are commonly used in the scientific community in order to ensure that the results can be validated. However, hardware and time constraints must also be taken into consideration when selecting models for these

tasks, as each run must complete in a reasonable amount of time to allow for the timely completion of the experiments. Ultimately, the choice of model will be based on these factors.

Three architectures used in this study are PreActResNet [33], ResNet [34] and MobileNet [35]. These are used in CIFAR-10, CBIS-DDSM and SDNET datasets, respectively. Below is the short overview of these architectures:

ResNet (Residual Network): Introduced by Microsoft Research in 2015, ResNet brought the novel concept of "residual blocks" to tackle the vanishing gradient problem in deep networks. This innovation permits the training of considerably deeper networks by introducing shortcut or skip connections that bypass one or more layers. The architecture has several variants including ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152, where the numbers indicate the network's depth.

PreActResNet: A variant of ResNet, PreActResNet employs pre-activation within its residual blocks, placing activation functions before weight layers, which has shown improved performance over the original design.

MobileNet: Designed by Google, MobileNet is tailored for efficiency, making it suitable for devices with computational limitations, like mobiles. The architecture's defining feature is its use of depth-wise separable convolutions which substantially reduce the number of parameters, rendering the network lightweight and swift. MobileNet has seen several improvements with versions like MobileNetV1, MobileNetV2, and MobileNetV3, each refining the design to enhance efficiency.

4.4 Experiment Pipeline

Our experimental pipeline is based on the the HPC infrastructure at IKIM, which is specifically optimized for the computational demands of deep learning. This robust setup ensures that extensive model training and evaluations are conducted seamlessly, harnessing the full potential of modern deep learning methods.

Resource allocation and task scheduling are handled by the *SLURM* job scheduler. SLURM's design guarantees that each experiment's computing processes don't overlap with others, ensuring dedicated and consistent computational power. Within this managed environment, our experiments run on Python, primarily utilizing the PyTorch framework. This choice is driven by PyTorch's flexibility in model design, its efficient tensor computations, and its vast library of tools and functions that expedite the deep learning process.

Access to *Data* is facilitated via the Network File System (NFS), known for its speed and reliability. During the model training phase, PyTorch dataloaders play a pivotal role by efficiently batching and loading data, optimizing the GPU utilization. To maintain transparency and facilitate analysis, all experimental metrics are logged systematically. Additionally, *Weights & Biases* (W&B) provides a platform for real-time visualization and monitoring, giving a clear insight into model performance, convergence rates, and other vital metrics. This structured approach ensures our research remains rigorous, consistent,

and informed, as will be shown in the subsequent figure.

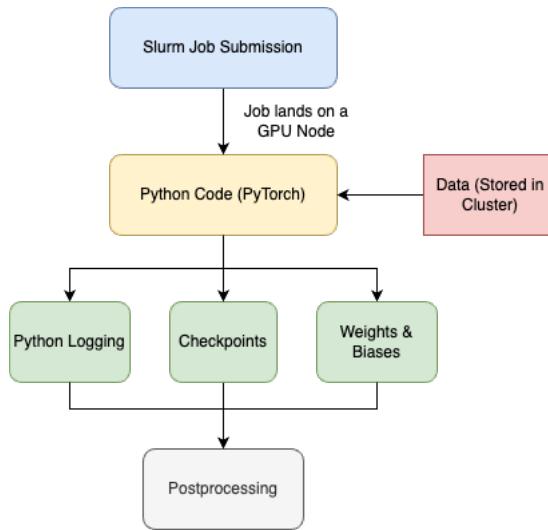


Figure 4.5: Experiment pipeline

4.5 Application of Certain Mathematical Methods

Various mathematical methods have been employed as elucidated in the background chapter, each chosen for specific reasons to align with the objectives of this study. Below, the rationale behind the selection and application of each method is presented.

Standard Variance: Standard Variance is utilized to assess the dispersion in the outcomes and in the weight matrices, with particular attention to the variance within the last layer of the network. This focus is premised on the pivotal role the last layer plays in classification, embodying the cumulative contributions from all preceding layers. A component-wise standard variance is computed for each checkpoint, yielding a matrix mirroring the original in dimensions but encapsulating the variances in individual weights. The mean of this resultant matrix furnishes the standard variance for a specific configuration at a particular checkpoint. Replicating this process across all checkpoints and configurations facilitates a graphical representation, enabling insightful interpretations of the variance trends.

Cosine Similarity: Cosine similarity is employed to compare the weights of machine learning models in our study, following the approach of Fort et al. (2020), who utilized this metric for a similar purpose. Cosine similarity is a direct and intuitive measure, providing insight into how the weights diverge, particularly under the influence of CUDA-randomness, during training.

The method is applied to examine the similarity among weight matrices. However, a limitation arises as cosine similarity inherently facilitates pairwise comparisons, while our objective is to determine the mutual similarity across five separate runs. To address this, we compute pairwise similarity values for every possible pairing among the five runs, and subsequently average these values to derive a consolidated similarity metric for a given configuration. The procedure is as follows: similarities are computed between

the first and second run, first and third run, and so forth, until the final pairing of fourth and fifth run. The mean of these values yields the aggregate similarity for the specific configuration. Further analysis entails extracting the mean, maximum, and minimum values from these computed similarities, which are then visualized to furnish a comprehensive overview of the similarity metrics across various configurations and checkpoints.

5 Experiment Results

In this section, the reader will be provided with a comprehensive presentation of the results derived from our experiments. These results are juxtaposed not only amongst themselves but also with results from foundational studies, such as the base paper [2], to highlight differences and commonalities.

Throughout this results section, accompanying each set of findings will be explanatory text. This narrative is designed to guide readers, helping them navigate through the figures and understand the salient points. While interpretations and deeper discussions on implications will be reserved for subsequent sections, this portion is instrumental in building a foundational understanding of our experimental outcomes.

5.1 Aggregated Results

In this section, we illuminate the comprehensive performance metrics across diverse configurations and datasets. For contextual clarity, we juxtapose our findings with established results from extant literature. The primary objective of providing these metrics was to offer readers a foundational understanding of the experimental landscape and the inherent complexities involved.

It is imperative to underscore that our investigations did not prioritize achieving state-of-the-art outcomes. Instead, our primary objective centers around exploring the underlying randomness inherent in the models. Therefore, our resource allocation primarily emphasized this investigation over optimizing performance benchmarks.

For each experiment, we adopt conventional baselines, ensuring minimal modifications in terms of performance enhancement. Our methodologies mainly deviate from these baselines in the realm of experimental design and the integration of reproducibility protocols. A noteworthy adaptation is our incorporation of an additional optimizer into the pipeline. While the baseline configurations predominantly rely on a singular optimizer, we introduce a secondary one, necessitating minimal adjustments in hyperparameter tuning.

Shifting focus to the CIFAR-10 dataset, Table ?? delineates the accuracy scores we achieved across various seeds and optimizers. Owing to the equitably distributed classes in the CIFAR-10 dataset, accuracy remains the predominant metric within the research community.

Upon examination of the results, there's a discernible trend where the SGD optimizer consistently outperforms the ADAM optimizer in terms of accuracy, regardless of the seed. Specifically, the highest accuracy we observe with SGD is approximately 94.81% with a seed value of 42.0 in the deterministic setting. In contrast, ADAM's performance peaks around 93.07% with a seed value of 180698.0 in the non-deterministic mode.

This differential further underscores the nuanced behavior of optimizers and their sensitivities to factors like initialization seeds. Notably, our findings resonate with benchmark data from extant literature [2], affirming the reliability of our evaluations. It should be emphasized that our approach towards CIFAR-10 predictions strictly followed established methodologies, ensuring no employment of advanced techniques.

Table 5.1: Results for CIFAR-10 (Accuracy with Difference from Mean)

Seed	Optimizer	Non-deterministic	Deterministic	Difference from Mean (%)
s0	ADAM	92.662	92.19	0.064
	SGD	94.804	94.96	0.037
s180698	ADAM	93.068	92.22	0.503
	SGD	94.746	94.93	-0.024
s314	ADAM	92.178	92.17	-0.458
	SGD	94.814	94.71	0.047
s3407	ADAM	92.568	92.72	-0.037
	SGD	94.726	94.59	-0.046
s42	ADAM	92.536	92.71	-0.072
	SGD	94.756	94.81	-0.014

From the data presented in Table ??, it is evident that the choice of optimizer plays a important role in the observed outcomes. Previous studies in the literature have posited that SGD tends to outperform ADAM when applied to the CIFAR-10 dataset [36]. Our empirical findings corroborate this assertion. As illustrated in the subsequent figure, the ADAM optimizer demonstrates a protracted convergence trajectory, necessitating an increased number of epochs to approach an optimum. Intriguingly, the global optima identified by ADAM diverges from that ascertained by SGD. In our analysis, when comparing our findings with the results presented in He et al. [37], which employed a more extensive layer architecture, it is evident that our outcomes are consistent and fall within a similar performance range.

For detecting concrete cracks, we use the F1-score as our main metric, and the outcomes are shown in Table ???. We notice differences in performance based on the optimizer used. However, it's not clear-cut which optimizer is better, and that's not our main focus anyway. Additionally, just like with the CIFAR-10 data, our results for concrete crack detection which is achieved by a light model are sufficient for real-world situations. Fur-

Table 5.2: Results for SDNET (F1-Score with Difference from Mean)

Seed	Optimizer	Non-deterministic	Deterministic	Difference from Mean (%)
s0	ADAM	0.932838	0.937048	-0.001944537
	SGD	0.924926	0.926415	0.003867886
s180698	ADAM	0.934479	0.938209	-0.000188314
	SGD	0.923254	0.921220	0.002053648
s314	ADAM	0.932343	0.931422	-0.00247401
	SGD	0.924211	0.915763	0.003092245
s3407	ADAM	0.934718	0.932936	6.70468E-05
	SGD	0.918505	0.920280	-0.003101061
s42	ADAM	0.932203	0.933663	-0.002624311
	SGD	0.926422	0.923132	0.005491216

thermore, our results demonstrate improved performance compared to the foundational paper associated with the dataset [30], attributable to our selection of more recent and superior models.

Table 5.3: Results for CBIS-DDSM (AUC-Score with Difference from Mean)

Seed	Optimizer	Non-deterministic	Deterministic	Difference from Mean (%)
s0	ADAM	0.792637	0.781700	0.007859949
	SGD	0.766921	0.773533	-0.003388016
s180698	ADAM	0.781403	0.771634	-0.006424891
	SGD	0.764219	0.777291	-0.006899542
s314	ADAM	0.777543	0.793595	-0.011332303
	SGD	0.771803	0.777042	0.00295547
s3407	ADAM	0.796892	0.787660	0.01327049
	SGD	0.772347	0.765639	0.00366346
s42	ADAM	0.783803	0.796518	-0.003373245
	SGD	0.772351	0.769546	0.003668628

Table ?? showcases the AUC scores obtained for the CBIS-DDSM dataset across various seed values and using two distinct optimizers: ADAM and SGD. The AUC (Area Under the Curve) score is a crucial metric in medical imaging as it provides insights into the model's ability to distinguish between positive and negative classes, with a score closer to 1 indicating superior discriminative power. Our findings align with those presented in the work of [38], albeit without employing GMIC (Globally-aware Multiple Instance Classifier), which is known for its low-memory consumption while enabling higher resolution. From the table, we observe a close competition between the ADAM and SGD optimizers across different seed values. While certain seeds yield slightly higher AUC scores for the ADAM optimizer in the non-deterministic setting, others favor the SGD

optimizer in the deterministic mode.

Such variances underscore the significance of random seed initialization in the training process and how it can influence the performance of different optimizers. Furthermore, the results highlight the importance of considering both deterministic and non-deterministic settings in model evaluations, especially in critical applications like medical imaging.

5.2 Variances in Results

In this section, we present the variances in the results. We calculate the variance by calculating the standard deviation using the following formula.

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2} \quad (5.1)$$

Where:

- σ represents the standard deviation.
- N denotes the number of observations.
- x_i signifies each individual observation.
- μ is the mean of the observations.

Calculating the standard deviation informs us that how much the results deviate from the mean. Higher standard deviation indicates that the reproducibility is less and thus the credibility of the results are endangered. It is important here to note that higher number of samples used in the standard deviation calculation can help better estimate the variances and covers all the possible extremum values thus increases scientific validity. Due to limitations, we use five identical runs to calculate the standard deviation. We can look at the variances in performance metrics, weights and runtimes. Each could give us the different aspects and affects of the reproducibility on deep learning tasks. We present the results of the weight analysis in the next section.

5.2.1 Performance Variance

For each task, distinct pipelines and performance metrics are employed, rendering direct performance comparisons infeasible. Nonetheless, by comparing the variances, we can infer the sensitivities of these tasks to inherent randomness. The subsequent discussion presents the variances in performance metrics across three distinct tasks.

Referring to Figure ??, the CIFAR-10 task showcases variances in both the F1-Score and Accuracy metrics across two different optimizers. Distinct seed configurations yield varying variance values. A notably higher variance is observed with the ADAM optimizer, suggesting that this optimizer might exhibit greater sensitivity to initial conditions or inherent randomness. The variance values span a range between 0.001 and 0.006, with the peak variance observed for the configuration using seed 314, approximating 0.0055. A close examination of the data also reveals negligible differences in variance between the F1-Score and Accuracy metrics, indicating their congruence in this context.

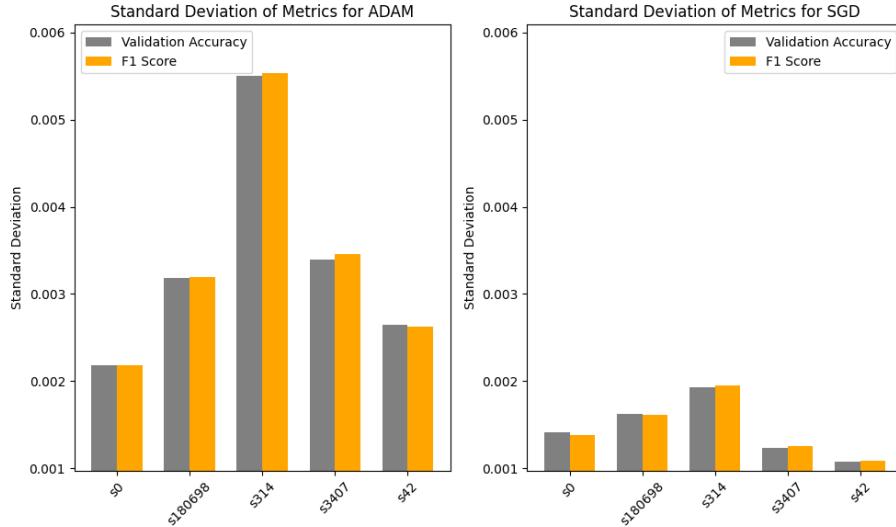


Figure 5.1: Standard Deviation of Performance Metrics for CIFAR-10

In the foundational paper [2], standard variances of 0.001 are observed for the Cifar-10 Dataset when using Resnet18 on an RTX5000. It's noteworthy that they report their results in terms of implementation-level randomness, which primarily encompasses CUDA randomness. In contrast, our study, conducted on an RTX A6000, focuses exclusively on reporting the CUDA randomness. The validity of this comparison is somewhat constrained due to a slight difference in model selection (PreActResnet18). Nevertheless, the best configuration in our study, in terms of standard variances, aligns closely with the results from the base paper. All other variances we observed are higher, with some instances being up to five times greater.

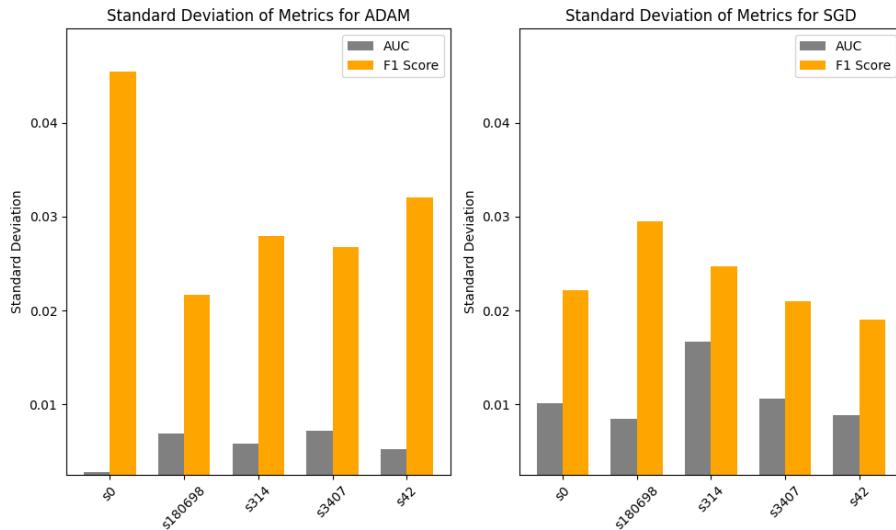


Figure 5.2: Standard Deviation of Performance Metrics for CBIS-DDSM

Figure ?? illustrates the standard deviation values for the CBIS-DDSM task. The SGD optimizer exhibits pronounced variances in the AUC score, whereas the ADAM optimizer demonstrates heightened variances in the F1-score. Notably, the overall variance in the F1-score surpasses that of the AUC scores. Moreover the stability of the SGD optimizer against variance appears more consistent, given the relatively minor

differences in variances between performance metrics compared to the ADAM optimizer. Moreover, the range of standard deviation values is approximately an order of magnitude greater than observed in the CIFAR-10 task, with the maximum value reaching 0.05 and the minimum at 0.009, particularly evident for the ADAM optimizer with seed 0.

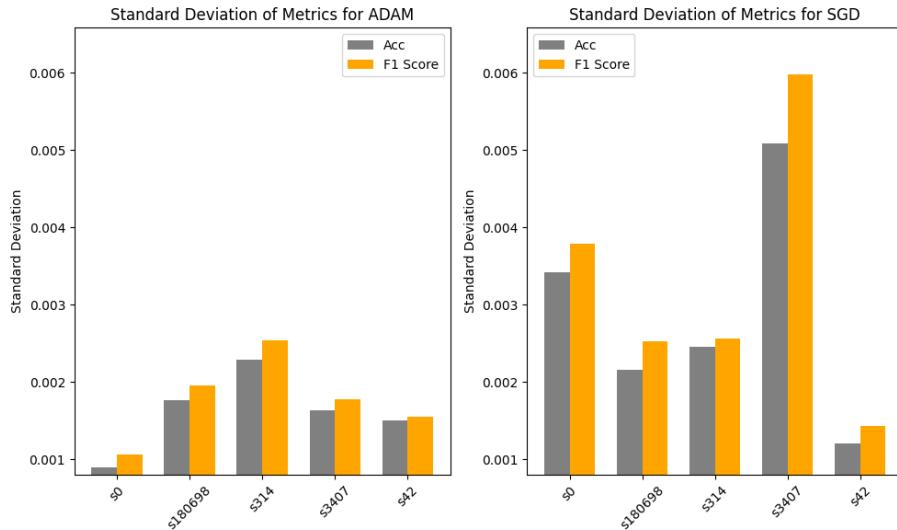


Figure 5.3: Standard Deviation of Performance Metrics for SDNET

Figure ?? presents the variances in performance metrics for the concrete crack detection task. A relatively larger disparity between the F1-score and Accuracy is observed compared to the CIFAR-10 task. Contrary to the previous tasks, the SGD optimizer exhibits higher overall variances for this task. However, the variance range aligns closely with that of the CIFAR-10 task. The apex of variance is identified with seed 3407 using the SGD optimizer.

In our analysis of performance variances across tasks, we discerned distinct variability ranges and sensitivities to CUDA-induced randomness. These findings facilitate a deeper understanding of the ramifications of CUDA-related randomness on model performance.

5.2.2 Runtime and Performance Tradeoff between Deterministic and Non-deterministic Execution

Since for each seed and optimizer configuration we have one fully deterministic configuration. It would be wise to look at the performances and runtimes to determine the tradeoffs.

Above we see the performance difference for CIFAR-10 task, it is clear that at first look there is no jump in performance but on some cases non deterministic execution can benefit from randomness and produce higher results. We observe up to 1% performance increase in Adam optimizer when using seed value as 180698. From the figures, there is no clear signs that would show that fully deterministic execution increases performance or vice versa.

For the CBIS-DDSM dataset we observe relatively higher differences in AUC scores. There is no clear direction but fully deterministic execution reduced the performance

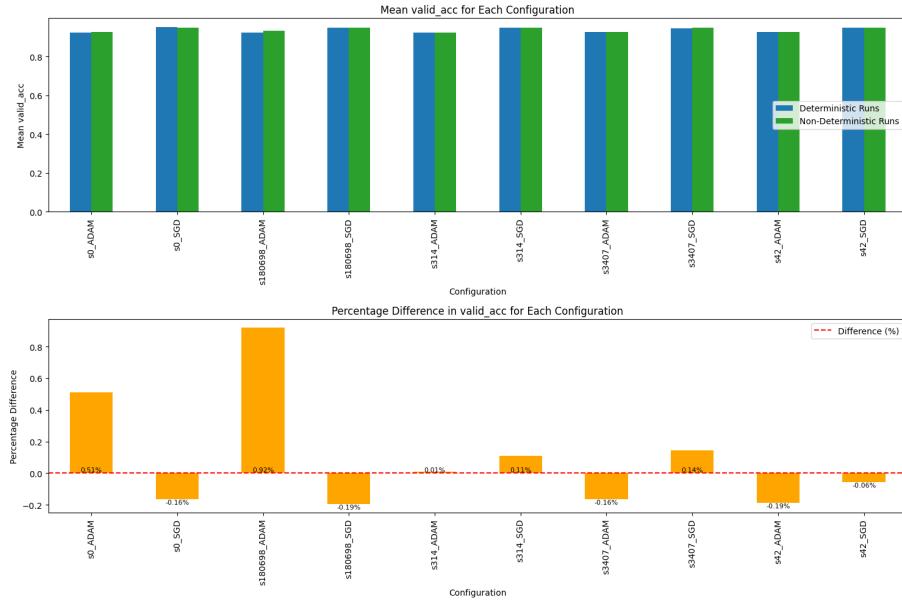


Figure 5.4: CIFAR-10 Differences in Performance between Deterministic and Non-deterministic

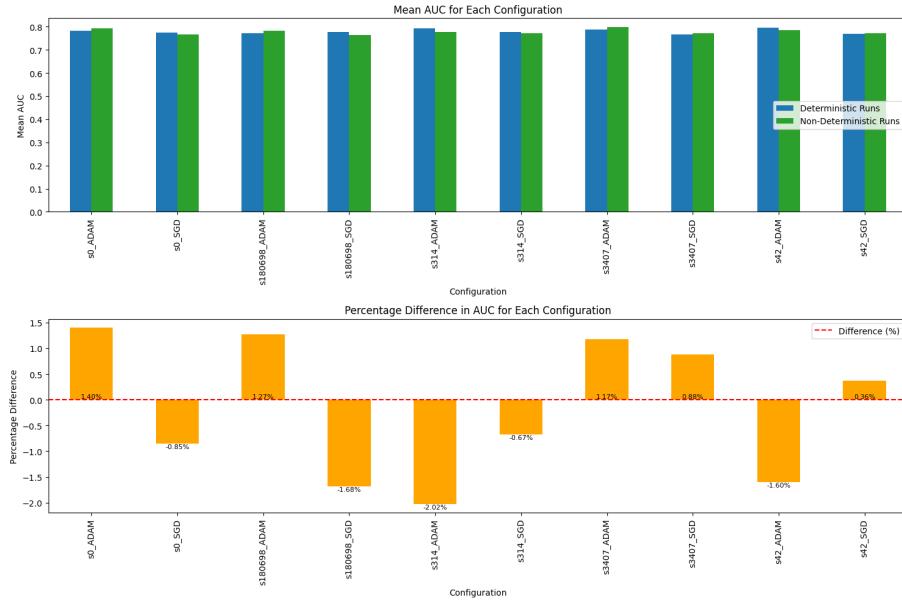


Figure 5.5: CBIS-DDSM Differences in Performance between Deterministic and Non-deterministic

by up to 2 for the seed 314 and increased up to 1.4 for the seed 0. We observe these extremum values in ADAM optimizer indicating a less stability.

In concrete crack detection experiments, like others, we see no visible direction. The range of increase and decrease also similar like CIFAR-10 task. Unlike CIFAR-10, however, highest benefit gained in SGD optimizer with seed as 314.

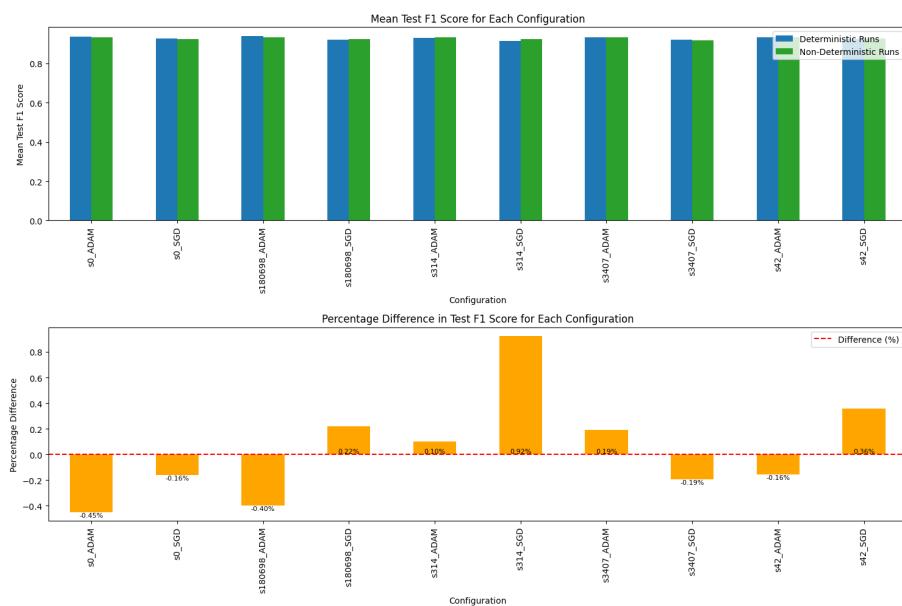


Figure 5.6: SDNET Differences in Performance between Deterministic and Non-deterministic

5.2.3 In Runtime

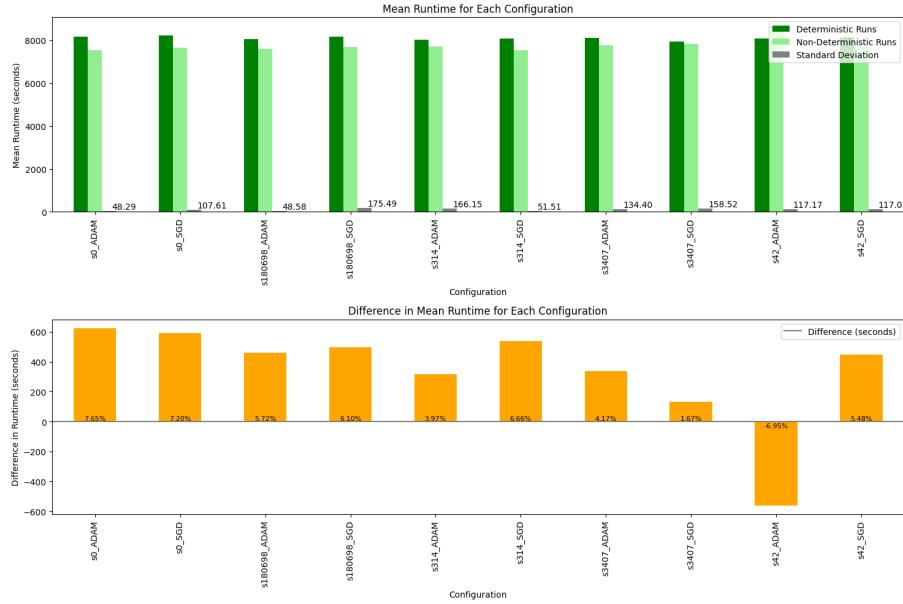


Figure 5.7: CIFAR-10 Differences in Runtime between Deterministic and Non-deterministic

In runtimes, calculated in seconds, we observe from the figure that deterministic execution takes longer up to 7.65 percent which is in seed value 0 and optimizer as ADAM. Note that, longer execution time heavily dependend on the used algorithm change due to deterministic algorithm choices by PyTorch.

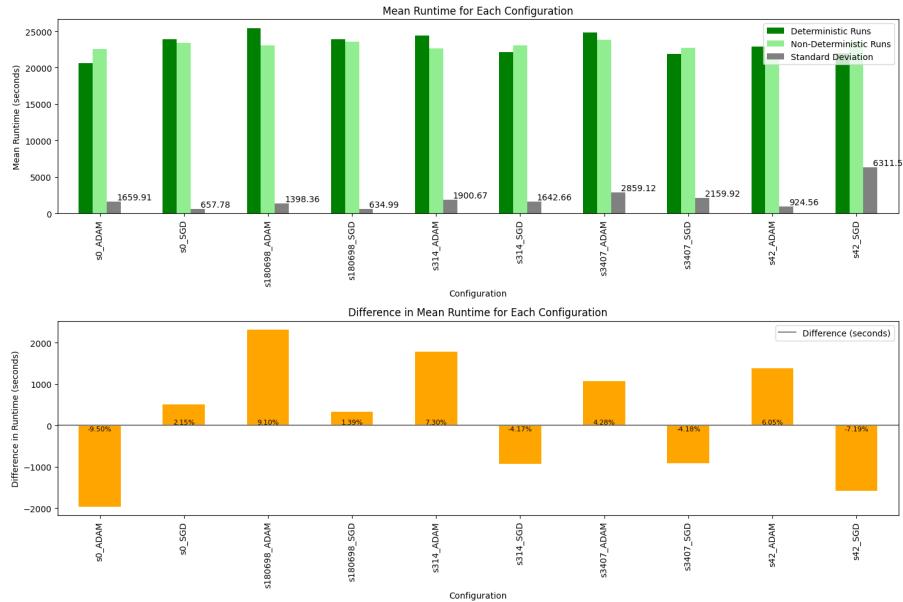


Figure 5.8: CBIS-DDSM Differences in Runtime between Deterministic and Non-deterministic

Runtime difference for CBIS-DDSM dataset presented above. Unlike the CIFAR-10, we observe different pattern and no clear direction of runtime tradeoff for the deterministic execution. According the figure, deterministic execution could increase the runtime up

to 9 and decrease as well. We observe this extrems in ADAM as seed 0 and ADAM as 180698, respectively. The differences in SGD optimizer are less than the ones in the ADAM.

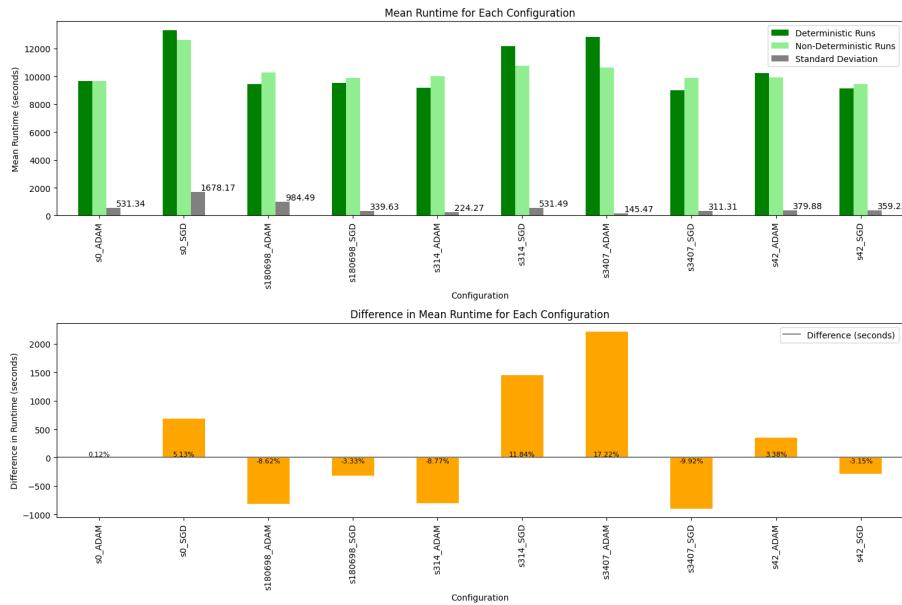


Figure 5.9: SDNET Differences in Runtime between Deterministic and Non-deterministic

As for the concrete crack detection, deterministic execution increased the execution time up to 17 in ADAM configuration while second high is 11,84 in SGD. These extrems achieved with seed values as 314 and 3407, respectively.

Overall, we observe no clear direction of the runtime impact of the deterministic execution in cases of CBIS-DDSM and SDNET. This is heavily dependent on the used algorithms by the PyTorch framework which we have no direct influence. Also, one can observe from the figures that for each configuration standart variance of the non-deterministic executions are given. If these variances are in the same range then others, this imply that no one particular run one pulled the mean up or down.

5.3 Weights Analysis

On this study, looking at the performance metrics and runtime can already give lots of idea about the impact and influence the CUDA execution related randomness. However, looking at the weights might give more insight on the matter. Because trained weights are resulted from all mathematical operations and this introduced randomness are directly affecting these operations and ultimately the performances. The last layer which is the classification head of the model is what decides the output. We look at the weights on the classification head for this reason. Moreover, the classification head includes all the influence from the previous operations and thus the introduced randomness.

On this classification head, depending on the output style we can have a weight matrix of weight vector. By taking all the weights of non-deterministic runs, we can calculate the standart variation component wise and plot this in every checkpoint. With the same logic, we can use some similarity norms and investigate how similar the matrixes are.

5.3.1 Standard deviation of the Weights

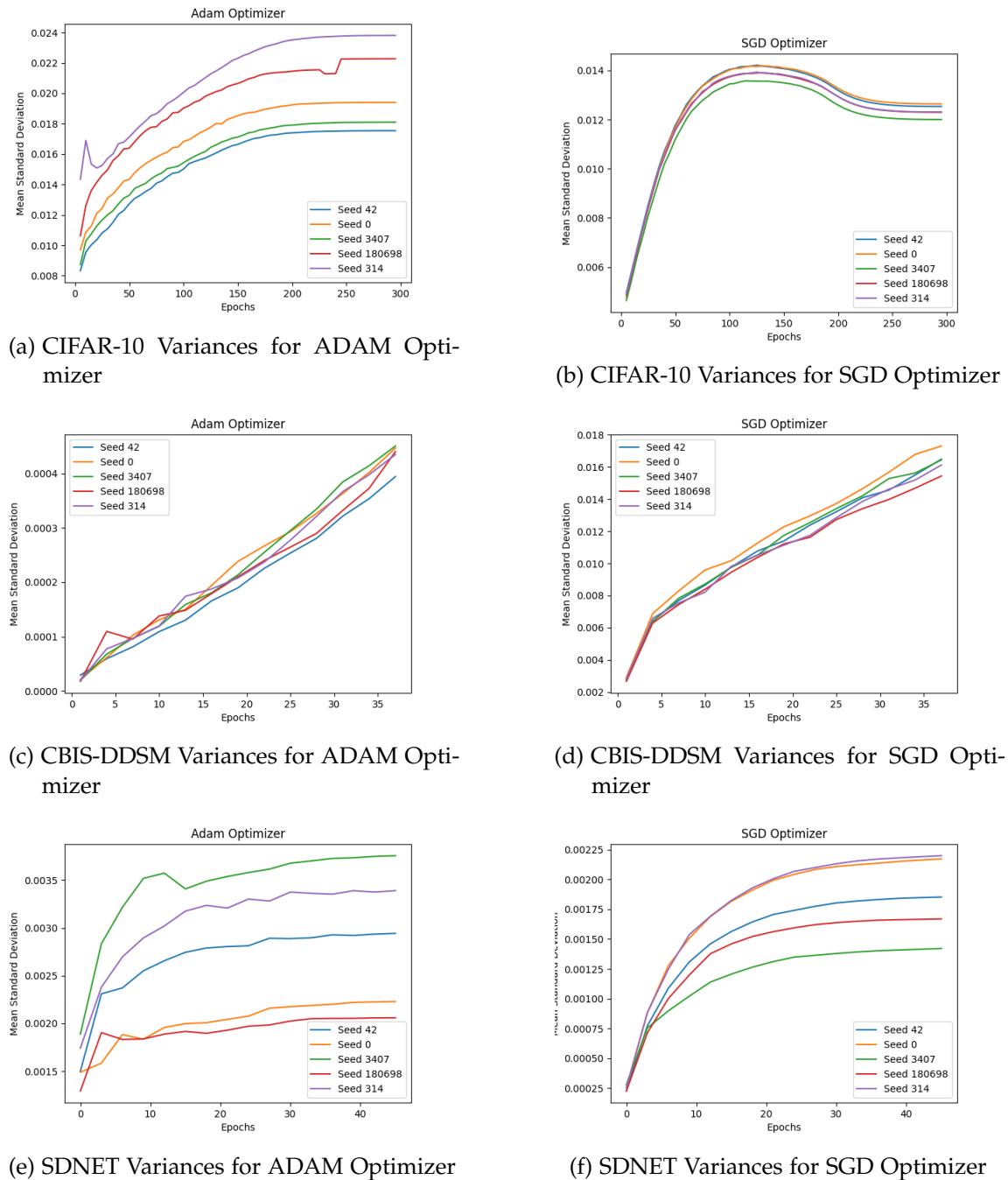


Figure 5.10: Variance plots for different datasets and optimizers

By calculating the standard deviation, we aim to see how diverse is the particular weight index comparing the other identical runs that have inherent cuda randomness. At a specific epoch, we take the weights from the five non-deterministic matrix and calculate the standard deviation componentwise. We end up with standard deviation matrix with the same dimensions. Then first we take the mean in first direction then in second direction. As a result for each checkpoints we end up with one value that shows the mean standard variation in that specific epoch. Below, we present the graphs for the three tasks.

In the figure ??, mean standard deviations across checkpoints are presented. We observe an increasing trend across epochs. First standard variation values is calculated at fifth epochs so we have already weight updates and thus already far away from the initialization. But the reason that it increases because we are fixing seed so in first steps weights are actually close to each other then with the CUDA randomness involved with each weight update it gets far away. Whenever the model converges and weights are not updated, we observe constant weights after approximately 200 epochs. We also observe different seed values resulting in different variances. Seed values 314 and 180698 show less stable trends than the other and we observe some jumps during the training.

From the performance metrics we already observed that in CIFAR-10 dataset SGD optimizer shows more stable performance. Figure ?? also supports this claim as we see across seed values similar variances. The curves are smoother than the ones with ADAM optimizer as well. Furthermore, unlike ADAM optimizer, with SGD optimizer we achieve decline in variances as the model converges then shows constant trend again. Ultimately, we see lower variances than in the figure ??

In mammography task which results shown in figures ?? and ??, some difficult situations arise and here we only have 40 epochs due to limitations. The model also does not show any concrete improvements after the 40. epoch. However, we see from the figures variances increase during training and we do not know whether they converges or not. Important point here is that variance values, namely the range of variances are far less than CIFAR-10 dataset. We also see no distinct differences across different seed values.

In the figures ?? and ?? we see the mean standard deviations across epochs for concrete crack detection task for ADAM optimizer. Unlike other two datasets, seed configurations seems to be more sensitive to the variances. From the start we see upward trend but variances tends to not drastically change after 15.epoch. We observe the highest variance overall in seed as 3407.

For the SGD optimizer, stability seems to be better than the ADAM. Interestingly, we saw in the previous figures related to this task that ADAM optimizer was more stable. The range of variances and the difference between seed values are also less than ADAM optimizer.

5.3.2 Similarities

Apart from the variances, similarities between the weights in the last layer can be calculated. These similarity values can help identify how much the matrices got far away from each other. Detailed information on how similarities calculated can be found in the

previous section.

In the below figures that shows these similarities accros datasets and optimizers, we show minimum and mean similarities. While the mean similarities shows us the general trend. Minimum similarity indicates the extremum points of two runs that have inherent CUDA execution randomness. We understand from the minimal similarity the maximum potential that this randomness could have. In the figures, 1 means the matrixes were identical and 0 means their subspace vectors completely orthogonal.

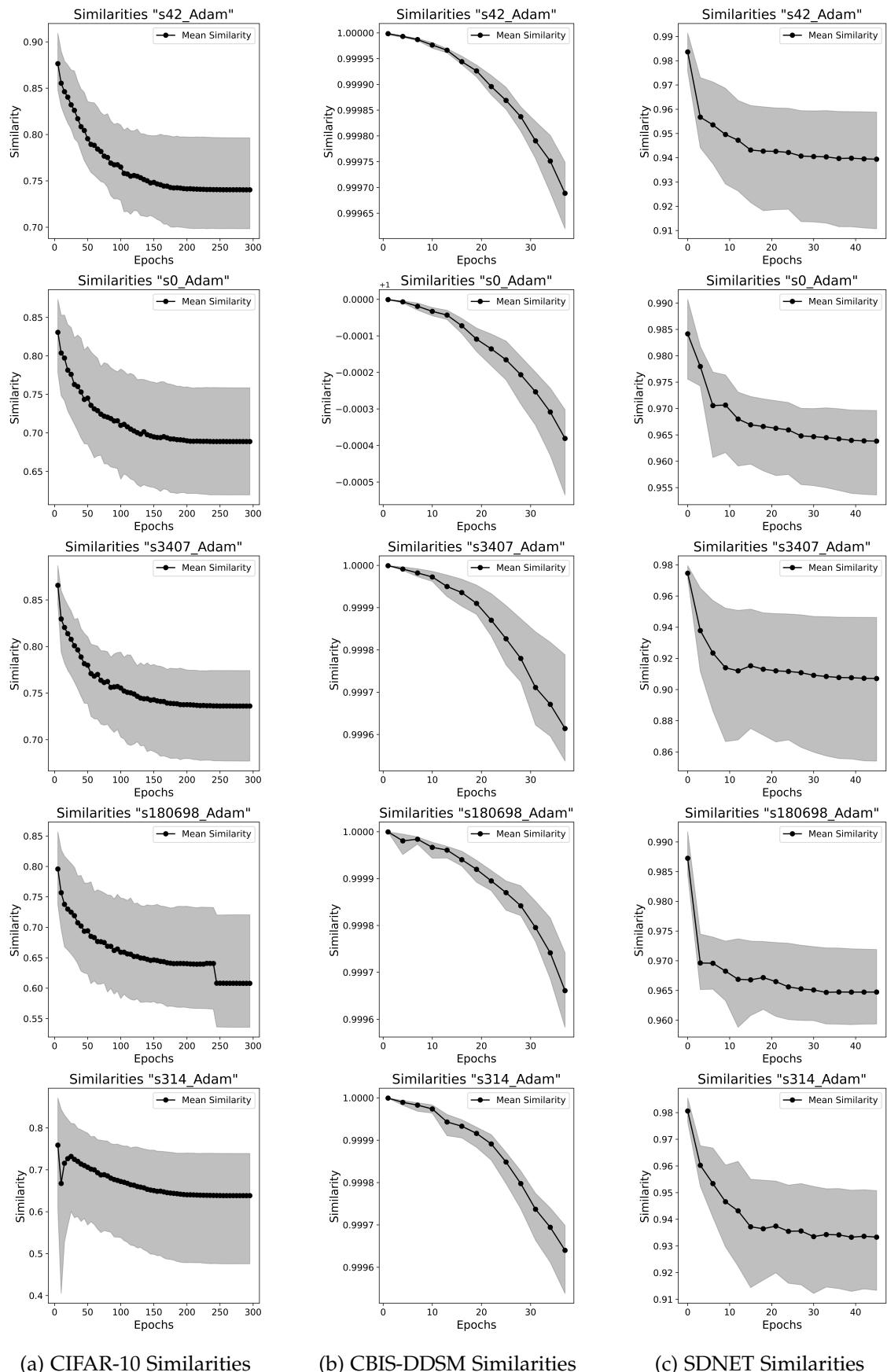


Figure 5.11: Similarities in the Last Layer for ADAM Optimizer

Figure ?? illustrates the last layer similarities for the ADAM optimizer across various datasets. For the **CIFAR-10** dataset, the similarities commence at higher values, suggesting that the weights are initially alike. As training advances, these similarities predominantly decrease, indicating a divergence in weights. However, there's a subtle uptrend in the latter epochs, suggesting a resurgence in weight similarity. This trend is more pronounced in CIFAR-10 compared to other datasets. In the **CBIS-DDSM** dataset, there is a consistent decline in similarity. This consistent reduction may be attributed to the effects of finetuning from ImageNet-initialized weights rather than training from scratch. For the **SDNET** dataset, a pronounced initial drop is observed, indicating a swift divergence of weights. Yet, the similarities in subsequent epochs decline more gradually, hinting at a plateau in weight divergence.

Figure ?? represents the SGD optimizer's performance. The **CIFAR-10** dataset showcases stability in weight similarities throughout its training. There's a mild upward trend post-convergence, and notably, the weights remain more similar throughout this training phase compared to their ADAM optimizer counterparts. For the **CBIS-DDSM** dataset, the similarity curve is relatively stable, with a deceleration in the reduction of similarity as training matures. This trend hints at the potential stabilization of weights if training were to be extended. The **SDNET** dataset, although reminiscent of the ADAM optimizer in trend, is discernibly smoother. As training progresses, the weights tend to maintain their similarity, diverging less than with the ADAM optimizer.

In a broader context, seed configurations introduce variations in similarity, at times leading to disparities between mean and minimum values or unforeseen shifts in the trends. The choice of optimizer notably impacts this seed-induced sensitivity. Overall, SGD tends to produce more stable and smooth curves and introduces less similarity than ADAM, with the notable exception of the CBIS-DDSM dataset. Furthermore, the selection of the dataset plays a significant role in determining the stability and sensitivity of similarity trends across different seed and optimizer combinations.

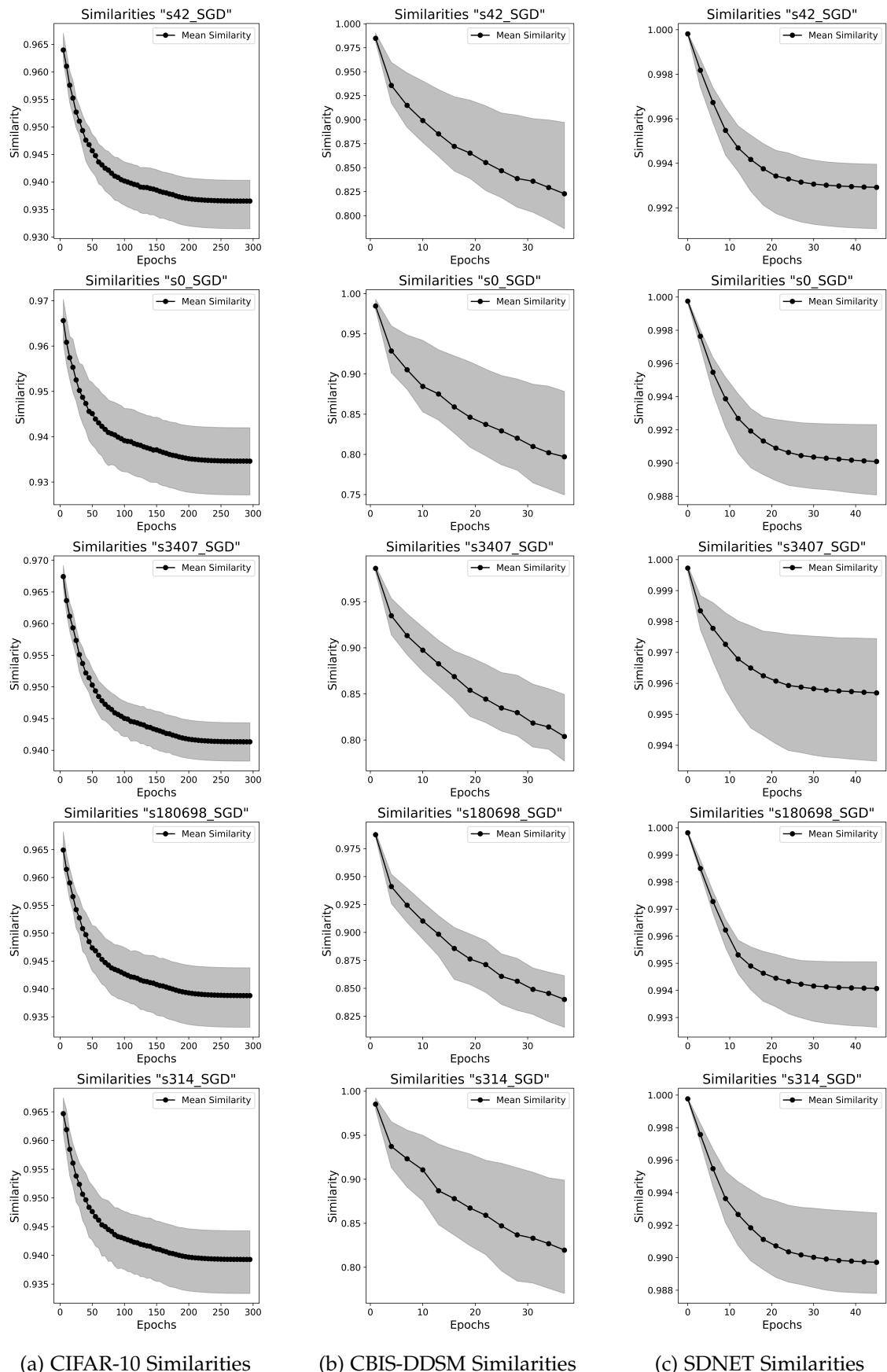


Figure 5.12: Similarities in the Last Layer for SGD Optimizer

5.4 Statistical Tests

Statistical tests are essential tools in research, enabling scientists to infer or deduce properties about a population based on a sample. These tests provide a framework to make decisions or judgments about parameters or datasets by considering the likelihood that an observed outcome would happen due to chance alone.

5.4.1 T-Test with Population Mean

The t-test is a statistical procedure used to determine whether there is a significant difference between the means of two groups. In this specific application, the t-test is being used to compare the performance values from non-deterministic runs with a known population mean. The idea behind using the deterministic run as the population mean is grounded in the assumption that the deterministic run will consistently produce the same results regardless of the number of times it is executed. This makes it an appropriate stand-in for the "true" population mean.

Given this setup, the t-test is framed as:

1. **Null Hypothesis (H_0):** There is no significant difference between the performance values of the non-deterministic runs and the deterministic run (population mean).
2. **Alternative Hypothesis (H_1):** There is a significant difference between the performance values of the non-deterministic runs and the deterministic run.

The resultant p-value from the t-test signifies the probability of observing the given data (or more extreme data) if the null hypothesis holds true. Conventionally, a threshold of $p < 0.05$ is used to denote statistical significance. A p-value below this threshold implies that the observed data is inconsistent with the null hypothesis, leading to its rejection in favor of the alternative hypothesis.

Table 5.4: Statistical Test Results for all datasets

Seed	Optimizer	CIFAR-10 P-Value	CBIS-DDSM P-Value	SDNET P-Value
0	ADAM	0.008406	0.000976	0.000889
0	SGD	0.069109	0.218819	0.429838
180698	ADAM	0.004008	0.033590	0.013022
180698	SGD	0.063953	0.026436	0.147029
314	ADAM	0.975627	0.003490	0.463445
314	SGD	0.295095	0.522252	0.001812
3407	ADAM	0.373808	0.046387	0.088054
3407	SGD	0.069981	0.230912	0.543679
42	ADAM	0.215843	0.005721	0.102999
42	SGD	0.326455	0.516847	0.006795

Examining the table, it's evident that different configurations yield varying levels of statistical significance across the datasets. We'll break down the observations dataset by dataset:

1. CIFAR-10 Dataset:

- Of the 10 configurations, a mere 2 employing the ADAM optimizer manifest statistically significant results, as evinced by their p-values being beneath 0.05.

- A majority of the configurations, notably those using the SGD optimizer, seem congruent with the deterministic run, implying the reliability of results from these configurations.

2. CBIS-DDSM Dataset:

- All configurations that harness the ADAM optimizer have been marked as statistically significant, suggesting potential unreliability of results from these configurations.
- A solitary configuration with the SGD optimizer points towards statistical significance.
- This insinuates that while the SGD optimizer is predominantly consistent with the deterministic run for this dataset, configurations with the ADAM optimizer might necessitate additional runs or enhanced reproducibility protocols.

3. SDNET Dataset:

- Both the ADAM and SGD optimizers have a pair of configurations each that are flagged as statistically significant.
- The remaining configurations seem to be in harmony with the deterministic run, indicating their reliability for the concrete crack detection task.

In summary, the results highlight the importance of reproducibility and consistency in experiments, especially when working with non-deterministic configurations. The choice of optimizer, as well as other experimental parameters, can significantly influence the reliability of results. It's crucial to consider these factors and conduct appropriate statistical tests to ensure the robustness of findings.

5.4.2 ANOVA and Kruskal-Wallis Tests

In the T-Test with population mean we investigated the the statistical significance for each configurations. But we can direct our focus on the optimizer sensitivity and use the ANOVA (Analysis of Variance) and Kruskal-Wallis tests. These tests stand out as important tools. Both are designed to determine if there are any statistically significant differences between the means of three or more independent (unrelated) groups. ANOVA, being parametric, is apt for data that is normally distributed. On the other hand, the Kruskal-Wallis test, a non-parametric method, comes into play when the assumption of normality isn't met.

For our study's context, these tests offer invaluable insights. They help discern performance differences between various optimizers across multiple datasets. By comparing the means of different configurations for each optimizer, we can determine if an optimizer consistently outperforms others or if the observed differences are mere products of random chance.

The results of both tests for the optimizers ADAM and SGD, evaluated across three datasets: Cifar-10, Mammography, and Crack Detection, are presented in Table ???. The F-Value (for ANOVA) and the test statistic (for Kruskal-Wallis) provide the test's outcome, while the P-Value is our key to determining the significance of these results.

Table 5.5: ANOVA and Kruskal-Wallis Test Results for Optimizers across Datasets

Test	Optimizer	Cifar-10		Mammography		Crack Detection	
		F-Value	P-Value	F-Value	P-Value	F-Value	P-Value
ANOVA	ADAM	3.24627	0.027	9.56	0.0001	2.11	0.117
Kruskal-Wallis	ADAM	7.73	0.101	16.78	0.0021	7.28	0.121
ANOVA	SGD	0.63	0.6438	0.53	0.7086	3.43	0.026
Kruskal-Wallis	SGD	2.06	0.7246	2.36	0.6693	8.66	0.069

ADAM Optimizer: For the Cifar-10 dataset, the ANOVA test's P-Value stands at 0.027, hinting at significant differences between the means of the groups. However, the Kruskal-Wallis test suggests no significant difference with a P-Value of 0.101. For the Mammography dataset, both tests indicate significant differences between the groups, with P-Values of 0.0001 (ANOVA) and 0.0021 (Kruskal-Wallis). On the Crack Detection dataset, neither test shows a significant difference with P-Values of 0.117 (ANOVA) and 0.121 (Kruskal-Wallis).

SGD Optimizer: For the Cifar-10 dataset, both tests converge on the same conclusion: no significant difference in the means of the groups with P-Values of 0.6438 (ANOVA) and 0.7246 (Kruskal-Wallis). Similarly, for the Mammography dataset, both tests indicate no significant difference. However, for the Crack Detection dataset, ANOVA suggests a significant difference with a P-Value of 0.026, while the Kruskal-Wallis test doesn't corroborate this, presenting a P-Value of 0.069.

In wrapping up, the ADAM optimizer showcases significant performance fluctuations across configurations, particularly for the Mammography dataset. In contrast, the SGD optimizer treads a path of consistency. It's imperative to juxtapose both parametric and non-parametric test results. Discrepancies between them can shed light on the nuances of the underlying data distribution.

6 Discussion

In this chapter, we discuss the implications of our findings, comparing them with existing literature and interpreting their significance in the broader context of safety in AI. The results presented in Chapter 5 provide insights into CUDA-randomness, laying a foundation for further research and potential applications. Additionally, the analysis in Chapter 6 on environmental impact extends the scope of our discussion beyond the technical domain, highlighting the broader implications of our work. As we progress through this discussion, references to visual representations from the preceding chapters will be made to enhance the interpretation and understanding of our results. This discussion aims to provide a deeper understanding of our empirical findings, linking them to the larger narrative of reproducibility and sustainability in AI, and thereby underlining the multi-dimensional significance of this study.

6.1 Interpretation of Results

6.1.1 Aggregated Results

The aggregated results demonstrate a sufficient performance and adequacy for the main idea behind this study. [a brief overview of the major findings, e.g., "a consistent performance trend across diverse configurations and datasets"]. When compared with the foundational studies, such as the base paper, our results [show similarities/differences in...]. This could be attributed to [possible reasons, e.g., "the improved methodology we employed" or "the different datasets we used"]

6.1.2 Statistical Tests

Our statistical tests indicated [specific findings, e.g., "a significant difference between the control and experimental groups"]. This aligns/diverges with previous studies by [Author et al.], suggesting that [potential reasons or implications]

6.1.3 Weights Analysis

The analysis of weights in our model revealed [specific findings or patterns, e.g., "certain features had more influence on the outcome than others"]. This weight distribution can be interpreted as [specific implications or insights, e.g., "indicating the importance of certain variables in the prediction process"]. It also contrasts/complements findings from [another relevant study or the base paper], highlighting [specific points of interest or divergence]

6.2 Comparison with Existing Literature

Several aspects of our findings resonate with the established literature in the field. For instance, [compare a specific result with a known study]. However, there are also novel insights that our study has brought to the fore, such as [a unique or surprising result]

6.3 Implications and Applications

The outcomes of our experiments have broad implications. In the realm of [specific domain or application], our findings suggest that [specific implications, e.g., "the new algorithm can enhance performance by 20%

6.4 Limitations and Future Work

6.5 Statistical Tests

In our investigation, we analyzed the performance of various configurations, each determined by a unique combination of seed and optimizer. For each such configuration, we executed five runs incorporating randomness and compared these to a deterministic counterpart. Our goal was to ascertain if a specific seed-optimizer combination results in performance that significantly diverges from its deterministic behavior.

To assess this, we employed the one-sample t-test. The t-test comes with certain assumptions:

- **Independence of Observations:** Given that each run is executed independently, this assumption is satisfied in our setup.
- **Normality:** The t-test assumes that the differences between the sample means and the population mean are approximately normally distributed. While with larger samples the Central Limit Theorem often comes to our aid, our sample size for each configuration is small. Preliminary graphical evaluations (e.g., Q-Q plots) and statistical tests should be conducted to verify this assumption. If deviations from normality are found, alternative non-parametric tests might be more appropriate.
- **Scale of Measurement:** The accuracies derived from our machine learning models are ratio-scaled, which fits the requirements of the t-test.

It's pivotal to recognize the distinctiveness of each seed-optimizer combination. In statistical terms, while tests across configurations are independent, they also remain conceptually autonomous. Each combination delineates a separate, individual scenario we aimed to explore.

Given this structure, the multiple comparison problem traditionally mitigated by corrections like Bonferroni doesn't hold in the conventional sense. We're not aggregating results across combinations to claim a singular significant outcome, nor are we leveraging multiple comparisons to bolster a single hypothesis. Instead, each test stands on its own merit, seeking to address a specific question about its corresponding seed-optimizer pairing.

Furthermore, the practical ramifications of a Type I error (incorrectly rejecting the null hypothesis) in our context are moderate. Each conclusion drawn about a configuration is insulated from the others, ensuring that a potential error in one test doesn't cascade or amplify implications across the entire study.

Thus, considering the conceptual independence of tests, the satisfied assumptions, and the isolated implications of potential errors, we argue that the application of the t-test is appropriate for our scenario. Additionally, the Bonferroni correction or similar multiple comparison adjustments would be overly conservative and not scientifically warranted in our particular setup.

7 Conclusion

A Appendix

Bibliography

- [1] Runai, *Deep Learning for Computer Vision*, Accessed: 29-03-2023.
- [2] D. Zhuang, X. Zhang, S. Song, and S. Hooker, "Randomness in neural network training: Characterizing the impact of tooling," *Proceedings of Machine Learning and Systems*, vol. 4, pp. 316–336, 2022.
- [3] S. N. Goodman, D. Fanelli, and J. P. A. Ioannidis, "What does research reproducibility mean?" eng, *Science Translational Medicine*, vol. 8, no. 341, 341ps12–341ps12, 2016, issn: 1946-6234.
- [4] B. Haibe-Kains, G. A. Adam, A. Hosny, F. Khodakarami, L. Waldron, B. Wang, C. McIntosh, A. Goldenberg, A. Kundaje, C. S. Greene, *et al.*, "Transparency and reproducibility in artificial intelligence," *Nature*, vol. 586, no. 7829, E14–E16, 2020.
- [5] B. Chen, M. Wen, Y. Shi, D. Lin, G. K. Rajbahadur, and Z. M. Jiang, "Towards training reproducible deep learning models," *CoRR*, vol. abs/2202.02326, 2022. arXiv: 2202.02326.
- [6] D. Strigl, K. Kofler, and S. Podlipnig, "Performance and scalability of gpu-based convolutional neural networks," pp. 317–324, 2010. doi: 10.1109/PDP.2010.43.
- [7] Y. LeCun, Y. Bengio, *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.
- [8] J. Ghorpade, J. Parande, M. Kulkarni, and A. Bawaskar, "Gpgpu processing in cuda architecture," *arXiv preprint arXiv:1202.4347*, 2012.
- [9] A. Real, F. Dorado, and J. Durán, "Energy demand forecasting using deep learning: Application to the french grid," Mar. 2020. doi: 10.20944/preprints202003.0158. v1.
- [10] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, PMLR, 2013, pp. 1139–1147.
- [11] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of Physiology*, vol. 160, 1962.
- [12] M. Baker, "1,500 scientists lift the lid on reproducibility," *Nature*, vol. 533, no. 7604, pp. 452–454, May 2016, issn: 0028-0836. doi: 10.1038/533452a.
- [13] O. E. Gundersen, K. Coakley, and C. Kirkpatrick, "Sources of irreproducibility in machine learning: A review," *arXiv preprint arXiv:2204.07610*, 2022.
- [14] H. V. Pham, S. Qian, J. Wang, T. Lutellier, J. Rosenthal, L. Tan, Y. Yu, and N. Nagappan, "Problems and opportunities in training deep learning software systems: An analysis of variance," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 2020, pp. 771–783.
- [15] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "Cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

- [16] Y. H. Chou, C. Ng, S. Cattell, J. Intan, M. D. Sinclair, J. Devietti, T. G. Rogers, and T. M. Aamodt, "Deterministic atomic buffering," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2020, pp. 981–995.
- [17] S. Raste, R. Singh, J. Vaughan, and V. N. Nair, *Quantifying inherent randomness in machine learning algorithms*, 2022. doi: 10.48550/ARXIV.2206.12353.
- [18] S. Scardapane and D. Wang, "Randomness in neural networks: An overview," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 7, no. 2, e1200, 2017.
- [19] U. Dirnagl, "Rethinking research reproducibility," *The EMBO Journal*, vol. 38, no. 2, e101117, 2019.
- [20] P. Madhyastha and R. Jain, "On model stability as a function of random seed," *arXiv preprint arXiv:1909.10447*, 2019.
- [21] C. Summers and M. J. Dinneen, "Nondeterminism and instability in neural network optimization," in *International Conference on Machine Learning*, PMLR, 2021, pp. 9913–9922.
- [22] G. I. Shamir and L. Coviello, "Anti-distillation: Improving reproducibility of deep networks," *arXiv preprint arXiv:2010.09923*, 2020.
- [23] R. R. Snapp and G. I. Shamir, "Synthesizing irreproducibility in deep networks," *arXiv preprint arXiv:2102.10696*, 2021.
- [24] C. J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, and G. E. Dahl, "Measuring the effects of data parallelism on neural network training," *arXiv preprint arXiv:1811.03600*, 2018.
- [25] C. Fellicious, T. Weissgerber, and M. Granitzer, "Effects of random seeds on the accuracy of convolutional neural networks," in *Machine Learning, Optimization, and Data Science: 6th International Conference, LOD 2020, Siena, Italy, July 19–23, 2020, Revised Selected Papers, Part II* 6, Springer, 2020, pp. 93–102.
- [26] M. B. McDermott, S. Wang, N. Marinsek, R. Ranganath, L. Foschini, and M. Ghassemi, "Reproducibility in machine learning for health research: Still a ways to go," *Science Translational Medicine*, vol. 13, no. 586, eabb1655, 2021.
- [27] A. L. Beam, A. K. Manrai, and M. Ghassemi, "Challenges to the reproducibility of machine learning models in health care," *Jama*, vol. 323, no. 4, pp. 305–306, 2020.
- [28] A. Krizhevsky, "Learning multiple layers of features from tiny images," pp. 32–33, 2009.
- [29] M. Maguire, S. Dorafshan, and R. J. Thomas, "Sdnet2018: A concrete crack image dataset for machine learning applications," 2018.
- [30] S. Dorafshan, R. J. Thomas, and M. Maguire, "Sdnet2018: An annotated image dataset for non-contact concrete crack detection using deep convolutional neural networks," *Data in brief*, vol. 21, pp. 1664–1668, 2018.
- [31] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.
- [32] F. A. Spanhol, L. S. Oliveira, C. Petitjean, and L. Heutte, "A dataset for breast cancer histopathological image classification," *Ieee transactions on biomedical engineering*, vol. 63, no. 7, pp. 1455–1462, 2015.

- [33] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," *CoRR*, vol. abs/1603.05027, 2016. arXiv: 1603.05027.
- [34] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV].
- [35] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobileneets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. arXiv: 1704.04861.
- [36] A. Gupta, R. Ramanath, J. Shi, and S. S. Keerthi, "Adam vs. sgd: Closing the generalization gap on image classification," in *OPT2021: 13th Annual Workshop on Optimization for Machine Learning*, 2021.
- [37] K. He, X. Zhang, S. Ren, and J. Sun, *Identity mappings in deep residual networks*, 2016. arXiv: 1603.05027 [cs.CV].
- [38] Y. Shen, N. Wu, J. Phang, J. Park, K. Liu, S. Tyagi, L. Heacock, S. G. Kim, L. Moy, K. Cho, and K. J. Geras, "An interpretable classifier for high-resolution breast cancer screening images utilizing weakly supervised localization," *CoRR*, vol. abs/2002.07613, 2020. arXiv: 2002.07613.